Information Systems 38 (2013) 155–181

Contents lists available at SciVerse ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

A unified description language for human to automated services

Daniel Oberle [a,*], Alistair Barros [b,1], Uwe Kylau [c,2], Steffen Heinzl [d,3]

[a] SAP Research Karlsruhe, Vincenz-Priessnitz-Str.1, 76131 Karlsruhe, Germany
[b] Queensland University of Technology, Level 3/126 Margaret Street, Brisbane, QLD 4000, Australia
[c] SAP Research Brisbane, Building A4, Level 7, 52 Merivale Street, South Brisbane, QLD 4101, Australia
[d] T-Systems International GmbH, Eigilstr. 2, 36043 Fulda, Germany

ARTICLE INFO

Article history:
Received 11 April 2012
Received in revised form
12 June 2012
Accepted 15 June 2012
Recommended by: D. Shasha
Available online 30 June 2012

Keywords:
Service
Service description language
Business networks
Service networks

ABSTRACT

Through the rise of cloud computing, on-demand applications, and business networks, services are increasingly being exposed and delivered on the Internet and through mobile communications. So far, services have mainly been described through technical interface descriptions. The description of business details, such as pricing, service-level, or licensing, has been neglected and is therefore hard to automatically process by service consumers. Also, third-party intermediaries, such as brokers, cloud providers, or channel partners, are interested in the business details in order to extend services and their delivery and, thus, further monetize services. In this paper, the constructivist design of the Unified Service Description Language (USDL), aimed at describing services across the human-to-automation continuum, is presented. The proposal of USDL follows well-defined requirements which are expressed against a common service discourse and synthesized from currently available service description efforts. USDL's concepts and modules are evaluated for their support of the different requirements and use cases.

1. Introduction

Through various technological developments in recent years, new and disruptive models have emerged that are accelerating wider and, increasingly, ubiquitous access to services. Software-, infrastructure-, and platform-as-a-service, business process outsourcing, cloud computing, service marketplaces, and service-centric business networks are a growing list of examples, where services are being commoditized, exposed and consumed beyond conventional boundaries. In addition to Web consumers, the reach extends to mainstream industries like transportation and logistics, banking and finances, public sector and manufacturing, when one considers the following sorts of Web services now available: track-and-trace of shipments, tariff look-ups, health insurance comparisons, smart home metering, business formation, ERP hosting, or electronic payments.

With the growth in number and sophistication of services widely available, the question turns to how effectively consumers can discover, understand and access services, without full reliance on providers. Experience so far has shown that any attempt to describe and catalogue services faces a common stumbling block: what is a service? Despite the widespread phenomenon of "service" in economic, political, business, communal and individual walks of life, there are still many uncertainties and tensions in developing a general conception of services that addresses the different notions and forms of services across different domains and applications.

* Corresponding author. Tel.: +49 6227 7 52585; fax: +49 6227 78 46758.
E-mail addresses: d.oberle@sap.com (D. Oberle), alistair.barros@qut.edu.au (A. Barros), uwe.kylau@sap.com (U. Kylau), steffen.heinzl@t-systems.com (S. Heinzl).
[1] Tel.: +61 7 3138 1199; fax: +61 7 3880762.
[2] Tel.: +61 7 3259 9629.
[3] Tel.: +49 661 896619; fax: +49 661 896649.

0306-4379/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2012.06.004

Following the prominence first given to services through Web services and service-oriented architectures, different efforts have approached the description of services from different objectives and perspectives. These have entailed ontology-based descriptions aimed at facilitating service discovery, composition and interoperability across otherwise technical descriptions of services through Semantic Web Services approaches, e.g., OWL-S [7] or WSMO [8] (allowing domain specific concepts to be annotated to service interfaces), and service reference models, e.g., OASIS SOA Reference Model [3] (supporting generic concepts). With the rise of software-as-a-service, where services are accessed as on-demand utilities, non-functional aspects, such as payments, legal obligations, geographic and temporal constraints of access, are becoming imperative, advancing service descriptions closer towards an understanding of services encountered in commercial practice (e.g., [17]). Other efforts have addressed the wider perspective of service networks, focused on value co-creation of services in partnerships and service bundling (e.g., Akkermans et al. [20] and De Kinderen and Gordijn [21]). Furthermore, service system efforts such as Alter [23] and Ferrario et al. [24] have aimed at systemic reference models across business and technical aspects.

As part of large research investments aimed at advancing service exploitation in large settings, under flagships such as the Internet of Services,[4] a new stage of maturity for service description languages has been sought. The *Unified Service Description Language* (*USDL*) [45] is the most prominent response to date. It has been developed across several research institutes and publically funded projects across Europe and Australia,[5] and this extends to the Americas as part of a standardization push.[6] USDL has been built and evaluated in a collaborative and interdisciplinary way where more than a dozen researchers have contributed by bringing in their expertise from different backgrounds (computer scientists, incl. security and SLA experts, business economists, legal scientists, etc.).

The overarching philosophy of development has been inspired from the design science approach [1] where previously developed service description concepts, languages, perspectives and experiences were harnessed, and USDL, at the outset, was situated at the *conceptual* level so that a variety of aspects could be analyzed without constraint of any one implementation language or technology. The kinds of services sought for support through for USDL included: purely human/professional (e.g., project management and consultancy), transactional (e.g., purchase order requisition), informational (e.g., spatial and demography look-ups), software component (e.g., software widgets for download), digital media (e.g., video clip players), platform (e.g., middleware services such as message store-forward) and infrastructure (e.g., CPU and storage services). Use cases from the corporate world provided insights into commercial management and arrangements of services such as cost centre ownership and provisioning, releasing and dependencies in complex IT and business landscapes. Use cases involving service marketplaces procuring services as complex as those from SAP's portfolio and ecosystem provided new insights into structures for service bundling of both professional and automated forms of services. Use cases from cloud computing/IT virtualization helped frame platform and infrastructure services into USDL and extended the notion of service dependency for complex configurations. Use cases from business networks have shown that service versioning/provisioning capabilities need to extend beyond service providers to intermediaries and outsourced players such as brokers, aggregators and channel partners – to drive up the "network effect" of services.

This paper sheds light on the constructivist development [34] of USDL, given the considerable developments that have taken place and the unprecedented variety of services aimed for USDL support. The paper starts off by surveying existing service descriptions efforts that have been the subject of significant research in service sciences and computing (Section 2). Section 3 continues by providing a consensual background of services in the socio-technical systems that they operate in by means of a universe of discourse. To guide the design of USDL concepts, language requirements encountered in other conceptual modeling languages are identified (Section 4). For the "grey area" of the suitability of concepts for services, particular requirements are proposed and framed against the service discourse. Section 5 provides an overview of USDL and explains how prominent non-functional aspects, such as pricing and legal constraints, lead to the split of USDL in nine "modules." The paper then discusses how the requirements have been addressed through its current state (Sections 6 and 7). Then, Section 8 discusses the methods of validation by means of a coverage analysis, i.e., it summarizes which constructs of USDL have been used in different case studies in diverse walks of life. Finally, Section 9 summarizes the paper and future development for USDL by way of a conclusion.

2. Related work

The following section provides an insight into state-of-the-art service languages, techniques, reference, and standardization efforts relevant for the conceptualization of service descriptions. These are grouped by means of their scope (strand), namely those concerning the purely technical considerations, functional meaning or semantics, more comprehensive descriptions factoring in business-level, non-functional properties, and dedicated service conceptions in wider systems or networks.[7]

The first developments of service languages concerns ***Service-Oriented Architectures*** (***SOA***). Web services are the predominant technical manifestation of services under SOA,

[4] Cf. www.internet-of-services.com

[5] Cf. the Role of the funding source section at the end of this document for a list of contributing projects.

[6] Cf. W3C Incubator Group on USDL at http://www.w3.org/2005/Incubator/usdl

[7] A noteworthy effort is the W3C *Service Modeling Language* (*SML*) [5]. SML is a strict superset of XML schema and does not contain any service related notion, however.

and XML is the common representation for Web services languages. Over the years, different standards bodies have specified several dozens of different aspects collectively known as *WS-**. The most prominent representative is the Web Services Description Language (WSDL), which provides a standard, XML-based definition of interfaces for software components implemented in different languages. Similarly, other WS-* specifications provide a platform-neutral focus on aspects such as security (WS-Security), policies (WS-Policy), or orchestration (WS-BPEL), to name but a few. Nowadays with the emergence of REST, the realization of SOAs is not confined to Web services based on WS-* specifications. As the number of collaborating entities in and across organizations, and also the number of the Web services, grows, a means of centrally discovering and gaining access to services is needed. A standard naming and directory specification has been defined for Web services through the OASIS standards body, namely *Universal Description*, *Discovery and Integration* (*UDDI*). It provides a framework for containing and linking basic organizational and technical information and thus realizes the function of white, yellow and green pages. However, UDDI prescribes only a tiny description scheme for services.

With the maturity of structural, behavior and access specifications for Web services, standards bodies have embarked in design and engineering considerations. In particular, the OMG standards body is developing the *Service-oriented architecture Modeling Language* (*SoaML*) specification [2] to support the activities of service modeling and design through a model-driven development approach. With the widespread acceptance of SOA and the plethora of its different conceptions and representations, a *Reference Model for Service-Oriented Architecture* (*SOA-RM*) from OASIS [3] has been proposed. SOA-RM captures key concepts and relationships concerning service behavior in a service-oriented environment. Similarly, The Open Group has defined an alternative reference model, viz., the *SOA Ontology* [4].

A second strand consists mainly of ontologies in the field of ***Semantic Web Services*** [6], which has proven a popular research field in recent years. The main goal is to capture the functionality of Web services in terms of application domains. For example, Web services can be conceptually described through logical capabilities having pre- and post-conditions. Descriptions of capabilities are then associated with WSDL operations of a Web service. The semantic annotation of Web services, as such, leads to the automation of discovery, composition, and invocation of Web services in a SOA by ontology reasoners and planning algorithms. The most prominent ontologies are *OWL-S* [7] and *WSMO* [8]. A plethora of surrounding and similar ontologies have emerged in the field. The W3C came up with a recommendation called *Semantic Annotations for WSDL* (*SAWSDL*) [9] to establish correspondences between tags in WSDL (or XML Schema documents) and concepts or relations in an arbitrary ontology. A similar idea is adopted by the W3C Member Submission called *Semantic Annotations for REST* (*SA-REST*) [10].

With the many approaches focused on semantics anchored to service operations, has come the need for understanding the relationship of services within their wider operational context. In this respect, OASIS is developing the *Reference Ontology for Semantic Service-Oriented Architectures* (*RO-SOA*) [11], an abstract framework for understanding significant concepts and relationships between them, within a semantically-enabled, service-oriented environment. In fact, RO-SOA builds on and formalizes SOA-RM. Current research topics in this strand tackle the concepts of Linked Open Data, RESTful services, and Linked Services [12], e.g., *WSMO-Lite* [13] or *MicroWSMO* [14].

In anchoring to SOA reference models as a way of providing a coherent semantic context for Web services, Semantic Web Services inherits a major restriction. The focus of SOA reference models, as observed above, remains on the operations capabilities of services, and the way request–response interactions take place between service consumers and providers. Such a restriction reflects the goals of automatic composition and invocation based on the semantics of Web services. Left by the wayside are significant aspects of services that one encounters in business practice which do not necessarily play a role in composition or invocation. Consequently, a third strand of service languages has focused on ***business*** aspects of services, and non-functional properties, as empirically derived from business practice and documentation concerning services. The German standard *DIN PAS 1018* [15] uses a business domain, as opposed to computationally amenable aspects of services, as the cognitive means of understanding services. In particular, it focuses on the tendering process and describes services without machine-readable concerns. The PhD thesis of Emmrich [16] has a similar motivation only that this work focuses on product-related services and its representation is structured in UML. This proposal merges existing standards and models for products, companies, organization, and resources. The PhD thesis of *O'Sullivan* [17] adopts a wider scope and contributes a domain-independent taxonomy that is capable of representing the non-functional properties of conventional, electronic and Web services. Toma [18] presents a syntactic translation of O'Sullivan's work in the proprietary WSML language. The goal is to extend the aforementioned WSMO by non-functional properties for automation of discovery, composition, invocation, and, in particular, ranking of services in a SOA.

Additionally, there are efforts that draw attention mainly to describing ***Service Networks***, i.e., the ecosystem and value chain relationships between services of economic value. An early work is Baida et al. [19] which is continued by the ontology of the *OBELIX* project by Akkermans et al. [20]. The latter is an application ontology that helps users with configuration of service bundling and graphical modeling of service networks. Similarly, the work presented by De Kinderen and Gordijn [21] introduces the *e^{3}Service* ontology to model services from the perspective of the user needs. This offers constructs for service marketing, but in a computational way, such that automated reasoning support can be developed to match consumer needs with IT services. The main focus of this work is to generate service bundles under the consideration of customer needs. Bitsaki et al. [22] introduce the *Service Network Notation* (*SNN*) which captures similar aspects to the e^{3}Service ontology.

Finally, there are overarching efforts that concentrate on the bigger picture of ***Service Systems*** also taking into account socio-economic aspects. These efforts mostly act as reference models leaving open a comprehensive conceptualization of non-functional properties of a service. Alter [23] was one of the first to realize that the concept of a service system is not well articulated in the service literature. Therefore, he contributes three informal frameworks as a first attempt to define the fundamentals of service systems. The work of Ferrario et al. [24] can be seen as a continuation and formalization of Alter's approach. Although differing in its main notions, they present *ontological foundations of service science* modeled according to the basic principles of ontological analysis. In turn, this reference ontology forms the core part of the *TEXO Service Ontology* [25]. In addition, it features ontology modules for pricing, legal, innovation, or rating information. Although the background of the *OASIS Reference Architecture Foundation for SOAs* [26] is service-oriented architectures, the specification argues that SOA-based systems are better thought of as ecosystems rather than stand-alone software products. Therefore, the specification is put into the service system category and takes the OASIS SOA-RM as its starting point by building on its vocabulary of important terms and concepts. Another effort considering the wider scope of the service system is the *Service Design Model* of Dhanesha et al. [27]. Other than the aforementioned efforts, the Service Design Model is geared at a software engineering purpose for the Eclipse Modeling Framework (EMF) [28]. The model takes into account the business organization, the customer, and the delivery organization during service design.

3. A discourse on services

The design rationale of USDL starts with a *"universe of discourse"* that ties together ideas and phenomenal features of services that have been elicited and generalized from the aforementioned languages, techniques, reference, and standardization efforts.

3.1. Services functionality

Services constitute encapsulated and exposed functionality, drawing from core artifacts, e.g., those related to business processes, applications, objects, and resources, constituting the enterprise phenomena surrounding services. Hence, services share similarities with these, such that they are sometimes regarded synonymously. This is especially the case with business processes whose gravitas is on the internal details of organizations and their systems, i.e., how requests, actions and responses are processed to fulfill consumer goals. In contrast, services are mostly targeted at external value and focus on interactions, between consumers, providers and others. Whereas business process activities are said to be orchestrated across collaborating resources, service capabilities are *delivered* to *consumers* by *providers*.

The foremost challenge of describing services is that they lie on a human-to-automation continuum. They range from purely human (e.g., consultancy services), to purely automated (e.g., message store/forward system), with a mixture of human and automated resources used for others in between (e.g., purchase order requisitions).

Across their diverse forms, services share a general characteristic. They provide functionality aimed at delivering value to consumers in terms of expected outcomes, subject to delivery constraints, e.g., availability, pricing, copyright or disclaimers. In doing so, they alleviate consumers with ownership of resources, costs or risks. Seen across practically all surveyed approaches, services involve active parts, for example, operations or actions, exposed to consumers, often referred to as *capabilities*. Thus, *service functionality* entails a set of capabilities, each generating value, measured in different ways, for expectations of consumers. Functionality can be captured in different layers, for different levels of concern. For instance, service providers can see the detailed functionality of services aligned to organizational resources and objects accessed (a "white-box" view). Intermediaries (cf. Section 3.2) can be limited to a less intimate view of functionality but sufficiently detailed so that they can configure third-party delivery functionality (a "grey-box" view). Consumers would only see a view of the service focused on their interactions (a "black-box" view).

Through the surveyed approaches and techniques, the functionality of a service is variously described as its interface and its capabilities are highly related to interactions that occur through these. The functionality (what), interface (where), and interaction (how) should be considered orthogonal when describing a service. Since the functionality of a service should in principle be reusable, it should be structured in a way that is configurable in different settings. Thus, variants of functionality should be supported through service descriptions and flexible use of concrete functionality should be supported through more abstract descriptions of service functionality.

3.2. Service agents and networks

With service delivery increasingly taking place in wider settings such as *business networks*, a distinction has emerged for service agents, e.g., owners of services and the providers who owners outsource core delivery responsibility to. In relation to this, the notion of *stakeholder* is important since certain partners in a business network have an interest or involvement in a service without having full responsibility for its provisioning and delivery. An example is a government regulation authority with a stake in compliance, such as customs and carbon footprint for commercial services.

Furthermore, the rise of service marketplaces, cloud computing, and business processing hubs are extending the distribution of provisioning to third parties. In service marketplaces, for example, providers from diverse agencies expose services, prices and conditions of delivery. The marketplace can then act as a broker between consumers and providers. Other *intermediaries* are emerging, e.g., cloud providers who can offer third-party hosting, or B2B gateways that can offer message translation as a service.

The surveyed SOA and Semantic Web Services approaches have anchored their conception of services around service capabilities between service consumer, broker, and provider

roles. Some of the business, service system and service network efforts consider more roles. However, they do not explicitly address these new trends of third-party service provisioning and intermediation that could be better supported in Web Services Architecture [48] and SOA platforms (e.g., see extended roles for service delivery in [29]). Proposals of the business strand have by far the strongest conceptions of services, however, they were developed before the onset of intermediaries.

3.3. Service dependencies

To provision a service requires the preparation of its necessary parts and supportive resources. For example, a purchase order service makes use of stock checking and order management services drawn from an enterprise solution. In turn, the order management service may be a composite service consisting of elementary services in order creation, modification, cancellation and shipment request. We can see from this example that services carry a structural *granularity*, classified in complex enterprise software solutions as core (elementary and typically not externally exposed capabilities), compound (capabilities relating to a single business object) and composite services (capabilities relating to multiple business objects).

In addition, there are different forms of service *dependencies* for different purposes (e.g., supportive functionality such as metering or billing engines). In complex IT landscapes with hundreds or thousands of services, there are often situations where services from different development backgrounds are not compatible with each other, e.g., because there is a lack of properly enforced governance. Vice versa, there are situations where services overlap in purpose and functionality because IT landscapes grow over many years (or even decades) out of multiple in-house developments and acquisitions. In some of these cases, they could be completely equivalent (same functional scope), whereas in other cases one service could substitute another (same or greater functional scope). The enterprise needs to be aware of such dependencies across application, platform and infrastructure levels, especially when services have to be substituted or packaged. Some of the dependencies are particularly explicit in the area of cloud provisioning [30]. In summary, it is necessary to extend the notion of dependencies from functional dependencies, i.e., the kind typically captured by surveyed efforts, to a more adequate notion that specifically indicates the different dependency types.

A particular and well-known manifestation of dependencies is *compositions* specifying data or control flow dependencies. It is interesting to note that despite the advances in service composition, the subject of considerable development under SOA, Semantic Web Services, business and service system efforts, the distinction between services vis-à-vis other enterprise artifacts is sometimes blurry. Compositions are not idiosyncratic to services but applicable to several kinds of enterprise artifacts such as business processes, workflows or software components. Another service-specific dependency is *bundling*. In contrast to composition, bundling defines no order but merely aggregates services in order to address marketing and revenue generations requirements.

3.4. Service delivery

The delivery of services is subject to constraints so that expectations of requests and responses are firmly in place for consumers, providers and intermediaries or stakeholders. Constraints include temporal availability, geospatial availability, dependent resources with necessary capabilities, and so on. Given the range and complexity of services across varying durations, the delivery of services entails rights, obligations, and penalties on the part of providers, consumers, and others. These are described in documents such as service contracts and service level agreements. The risks of contention, negligence or malpractice are considered significant. Hence, non-functional properties need to be made explicit for service delivery. This is in contrast to business process orchestration where such information is regarded as tacit. To date, the work of O'Sullivan [17] has been one of the most comprehensive proposals on the conceptual level for business-focused, non-functional properties of services. However, O'Sullivan's conceptual model has not been validated for service networks under the emergence of intermediaries.

One of the most significant delivery constraints is the pricing of a service whose key challenge is the variety of conditions contributed by different times, different customer profiles, and different industries. In pricing theory, this is referred to as the *segmentation* of pricing, i.e., the rules governing when and how different consumers are charged different prices [36]. Service pricing typically features a hierarchical structure where a price plan is made up of several price components. This static structure is often complemented by *price fences* that capture the dynamic variations on pricing such as the rewards status of the customer.

Delivery constraints under distributed service provisioning (i.e., a channel partner exposes a service through a marketplace, whose hosting parts are federated across a cloud provider) are incrementally built up over the delivery chain of contributing partners. By implication, pricing and other delivery constraints are relative to other parties in the delivery chain. For example, the price of channeling is a portion of the overall price, although the delivery needs the broker and hosting services in place. The (true) price for the consumer of the service involves prices of the different partner services. Such constraints in complex delivery chains are not apparent in any of the surveyed efforts.

3.5. Service consumption

Services are accessed through designated points of consumption known as *channels*, i.e., forms of business resources which have consuming applications that allow access to or specific operations of services. For end consumer interactions, different operations of a service may be required in different channels. For example, a passport application requires full authentication of a consumer through a counter channel. Progress in processing the passport may involve a

self-serve channel while final handover of the passport can only occur through the counter. Channels can support different technical channels for different types of devices that are permitted when interacting with services.

Existing SOA and Semantic Web Services efforts have the notion of the external behavior of services although they do not explicitly support channel constraints for particular capabilities, e.g., which capabilities require human interaction for full authentication and which may be accessed through mobile devices. Business efforts address service consumption requirements as part of non-functional properties of service delivery (e.g., the geospatial and temporal constraints of where services are consumed). However, they do not make reference to an explicit behavioral view of a service upon which its functionality, resources and constraints of a service can be suitably comprehended by consumers (e.g., external service model [31]).

4. Language requirements for USDL

Based on the general discourse of services described above, the following section identifies key requirements to guide the formation of service description languages. These requirements are split into generic language requirements and service-specific concept formation requirements.

4.1. Generic language requirements

First and foremost, a comprehensive description of services, across all the facets and perspectives of the wide variety of stakeholders, is best supported on a conceptual level [32], i.e., the well-known *conceptualization* requirement. Correspondingly, a service description language should have a sufficient *expressive power* so that full conceptualization is possible. Given the size and complexity of service related information, a service description language should explicitly support *modularity* in order to improve maintainability of service descriptions. Given the diverse industries and domains involving services, a "one-size-fits-all" service description scheme is not feasible. Even on a generic level for aspects such as service ownership, pricing and availability, new and unforeseen requirements for descriptions should be expected. Hence service description languages should support *extensibility* so that they can be used in specific business contexts involving new requirements that have not been factored into the supported set of service descriptions. In light of the size and complexity of service related information, and the fact that service domain experts are non-technicians, service descriptions languages should provide *comprehensibility*. As with languages used for other applications, it is crucial that a service description language has a *formal foundation*.

4.2. Service concept formation requirements

In information systems conceptual modeling literature, the requirement of the *suitability* of a language or technique for its intended domain is prominent. Different domains require different language concepts and features suitable for them. The issue of suitability gets to the heart of a *constructivist* synthesis of concept formation for a service description language. In [33], principles were proposed to guide the identification of suitable modeling concepts for business processes. Since business processes are closely related to services (cf. Section 3.2), these requirements are adapted for the suitability of service description languages in the following.

4.2.1. Organizational embedding

In reality, services are organizational systems or subsystems and so all concepts of services relate, directly or indirectly, to organizational concepts. In that sense, the concepts, as with those of other types of organizational artifacts, such as business processes, objects and resources, are said to be *organizationally embedded* [34]. More specifically, services are grounded in organizational functional structures. The requirement entails alignment of service descriptions with organizational concepts, whether they reference organizational concepts elsewhere maintained (e.g., LDAP directories) or whether they provide corresponding concepts.

Fig. 1 provides an abstract illustration of organizational embedding. The middle shows an abstract concept dependency structure of an organization. At the business level, this could consist of the functional structure, broken down into organizational capabilities and, in turn, artifacts (resources, products, services processes). At the IT level, IT automated services, applications, platforms and infrastructure would be incorporated. A service could be at the business or IT level, and its dependent organizational artifacts across business and IT levels would be traceable. The figure shows a service, whose details are stored in a service directory, and other resources supporting techniques or methods capturing models related to the organizational embedding, at different levels.

As apparent from Fig. 1, there are several key implications for the design of service description languages arising out of organizational embedding. First, concepts can be justified in terms of wider systemic phenomena (e.g., policies or goals of the enterprise or its units). Second, the languages can be aligned with other languages, techniques or methods also concerned with organizational systems, i.e., service description languages do not override the concerns of these other languages, e.g., service description languages relate to, and should not be confused with, business process modeling and organizational resource languages. Third, service description languages can be used in larger organizational methods such as enterprise architecture frameworks where they can be co-opted with wider operational, tactical and strategic concerns for organizations, e.g., strategic planning or marketing. This is in line with the proposals of the business approaches.

4.2.2. Cognitive sufficiency

Barros and ter Hofstede [33] identified the need for business process models to provide a sufficient cognizance for human interpretation. This need led to the requirement for integrating information about control and data flow, resources, etc., through models that reduce the need for assumptions to be made due to missing details. Accordingly, we require that practitioners reliably

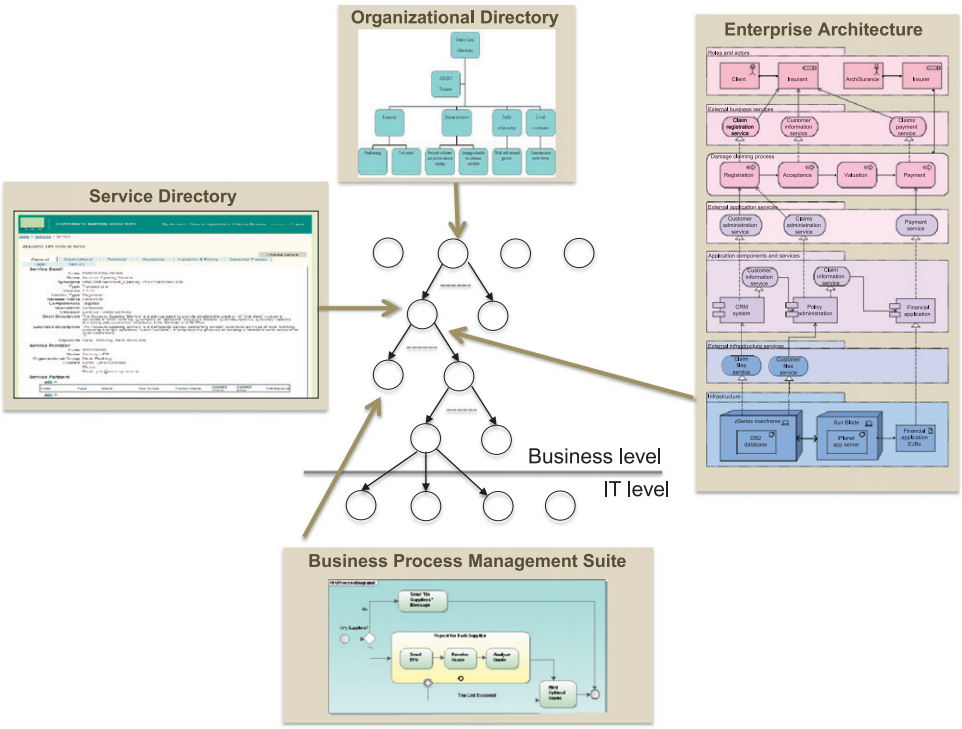

Fig. 1. Organizational embedding.

understand the different aspects of services including models of their different artifacts, documents, code, and non-functional descriptions.

Fig. 2 illustrates the cognitive sufficiency requirement for a service across different representations of its details, attributes, models and documents. The general details of a service, on the left hand side, lead to the details of how the service is delivered from the perspective of a consumer, on the top. Interestingly, this part combines attribute descriptions of the different phases of delivery and a model diagram. The details of service level agreement (SLA), on the right, can be viewed for the service as a whole. However, individual operations (their inputs, outputs and channels where they are accessed), seen at the bottom, can be used to highlight different parts of the SLA document related to that operation.

Concluding, a service description language should abstract from the core functionality of a service (capabilities and operations) through its descriptions. It is then possible to provide more fine-grained correlation with different details of service, allowing delivery constraints of SLAs, pricing etc. to be referred to. The intuitive meaning of a service is then conveyed coherently without requiring practitioners to make mental correlations across different service documents.

4.2.3. Service information hiding

The well-known principle of *Information Hiding* is intrinsic to most services since they are encapsulated, reusable and deployed units of functionality. Current service languages support well-defined service interfaces while current service composition languages allow services composed on business process (e.g., BPMN), service operation (e.g., SCA) and UI levels (e.g., widget composition) to be hidden behind interfaces. However, services have complex structures and can recursively consist of other services of different sorts and with different types of artifacts.

Fig. 3 shows services descriptions captured for service records maintained in a service directory, on the left. The service records contain references to external files containing models of different artifacts related to services—correspondingly, those artifacts are encapsulated as services. There are three artifacts. A service bundle, on the middle top, combines an iPhone and a business process. The iPhone is catalogued through a digital asset management repository. The business process used in the bundle, in the centre, is also exposed as a service. This is indicated by the service directory. The business process has a model managed through the business process management suite. An individual activity in the business process makes use of a composed service, on the bottom, maintained through a service component architecture tool. The composed service is also maintained in the service directory, however note, its constituent services are not explicitly catalogued. Taken together, the service directory makes the services across different artifacts and business operations, and keeps track of their dependencies, thus promoting flexible reuse.

4.2.4. Deployment symmetry

As services are extended to third parties, such as brokers or cloud hosters, there should be no loss of information in terms of their functional and non-functional aspects. As extensions are created, services should be traced back to their previous versions and extensions should occur with

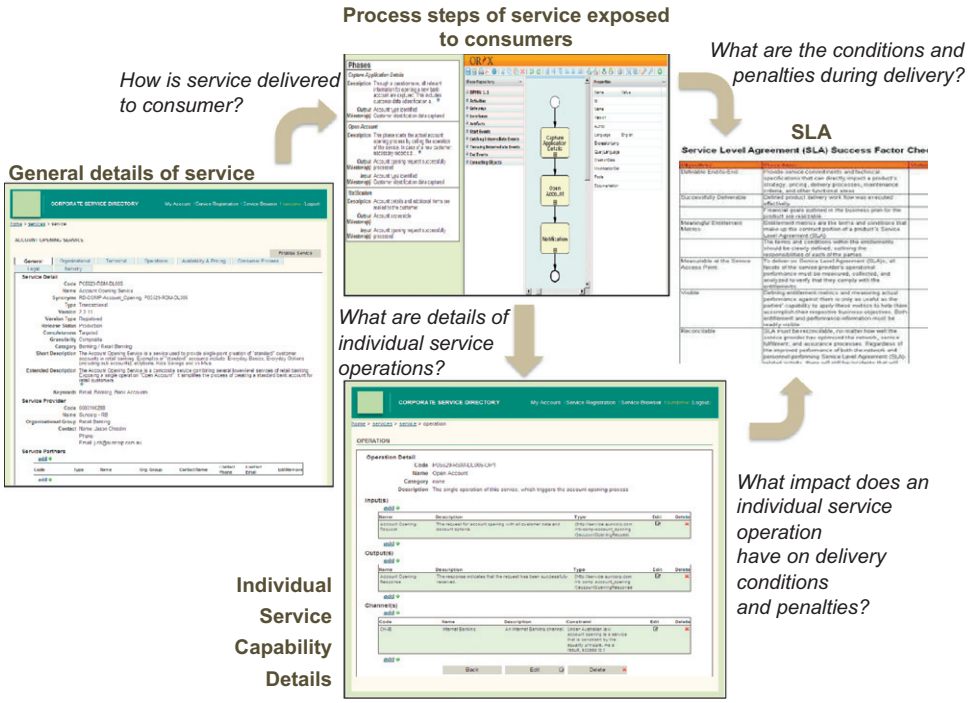

Fig. 2. Cognitive sufficiency.

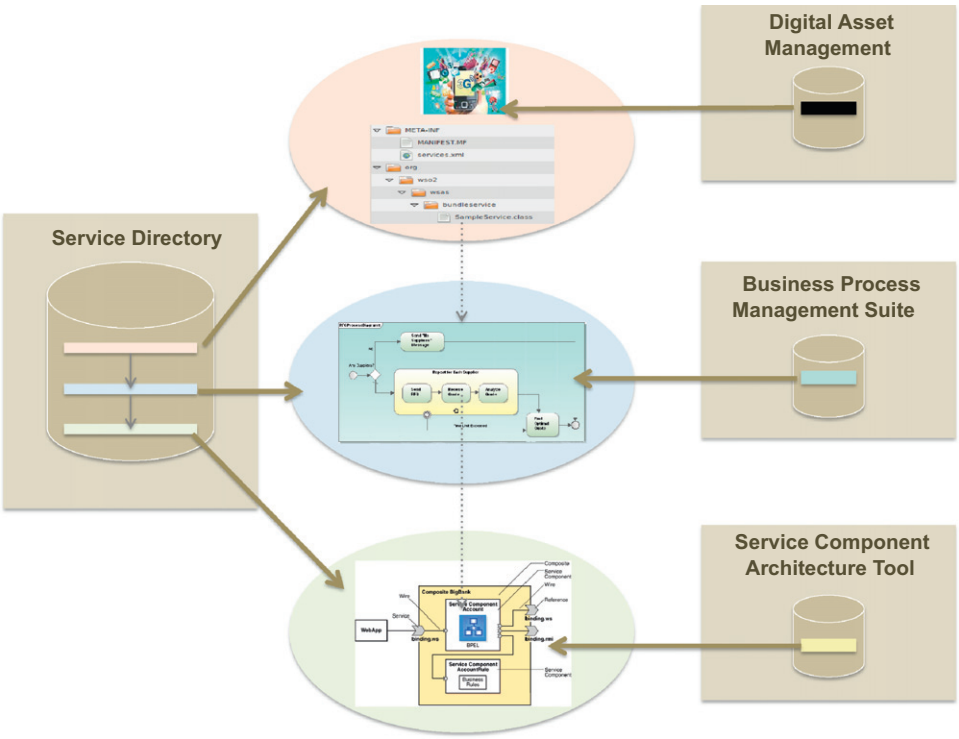

Fig. 3. Service information hiding.

reference to prior constraints and core service details. Therefore, languages should allow service descriptions to be consistent and coherent, i.e., *symmetric*, regardless of how they have been extended and where they have been deployed. Deployment symmetry has not been addressed by any of the surveyed efforts.

An implication of this requirement concerns the organizational context of services. Fig. 4 shows that a service

owner advertises a service together with the core part of its "org. structure" through a service broker. The "org. structure" consists of designated roles required for undertaking the service as policies, for the purposes of regulatory compliance (e.g., with a government). The service can then be ordered through the broker with a target organization setting needing to comply with the constraints set in place by the service owner. The new setting distributes the required delivery through different services providers (i.e., in a virtual enterprise). Irrespective of whether the service is run through its originating operational environment or new ones, the service descriptions set in place contain constraints so that service delivery conforms to required constraints. Therefore, the context and structure of services should be defined *abstractly* so that concrete bindings can be configured in particular specializations of services.

4.2.5. Execution resilience

Functional descriptions of services generally describe what is entailed in the delivery; another critical concern for ensuring that a service is deliverable is exception handling. The natural place for exception handling is in detailed specifications of service tasks, e.g., in pre- and post-conditions and action specifications. Therefore, a service description language should support the handling of exceptions, so that the execution resulting from a specification can be validated as being resilient. This includes rollbacks, cancellation policies, contingencies etc. These issues, of course, relate back to other non-functional issues. For example, "cooling off" periods can be built into pricing should there be uncertainty about customer satisfaction concerning the fulfillment of a service. In addition, service level agreements need to correlate with exceptions so that compliance of delivery can be assessed when things go wrong.

The requirement of execution resilience has been addressed in service/process orchestration and composition languages, such as WS-BPEL, BPMN and SCA, through technical exception handling considerations. However, exceptional issues at the business level, such as an explicit support for policies for cancellations and contingencies are only dealt with implicitly through SLA languages. A comprehensive treatment is not available through surveyed efforts.

5. Design of USDL

Given the complexity of the service domain, seen from the discourse in Section 3, USDL has been designed with the requirements of *conceptualization* and *modularity* in mind. With respect to the former, UML class models are used for capturing USDL via the Eclipse Modeling Framework (EMF) [28].

With respect to modularity, USDL is split into several packages (according to UML terminology). Each package represents one USDL "module" and contains one class model. The resulting split in modules follows from prominent non-functional aspects such as pricing and legal constraints, how services are interfaced with for delivery and service level agreements, which partners have responsibility for the service and details about service functionality. The nine modules are shown in Fig. 5 and briefly explained in the following (classes and relations are written in **bold** font). Note that it is beyond the scope of this paper to depict every class diagram. Instead, detailed specifications of each module are available at [35].

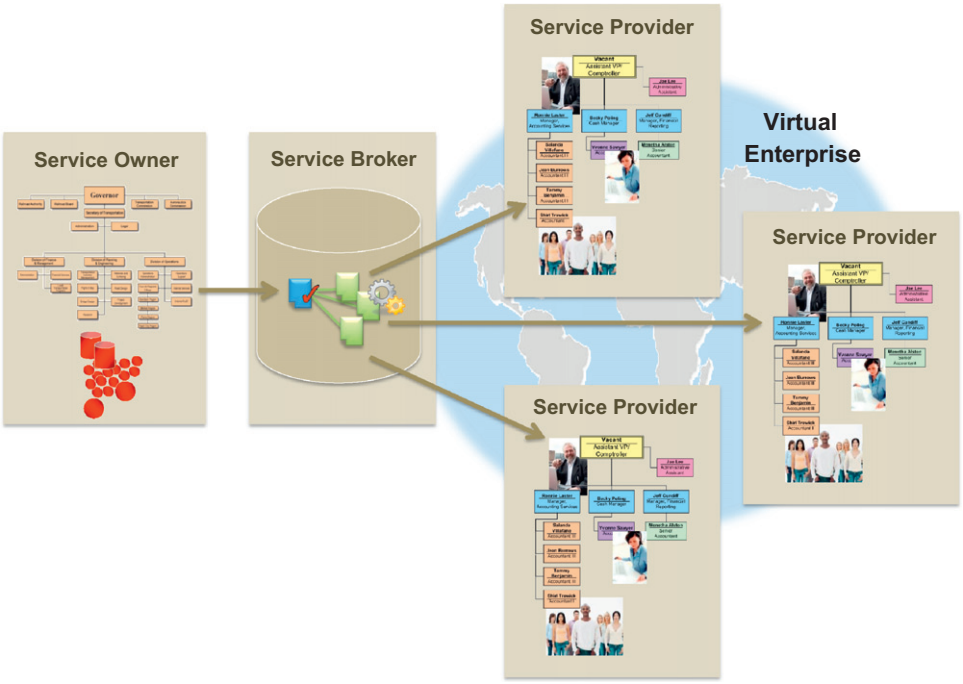

Fig. 4. Deployment symmetry.

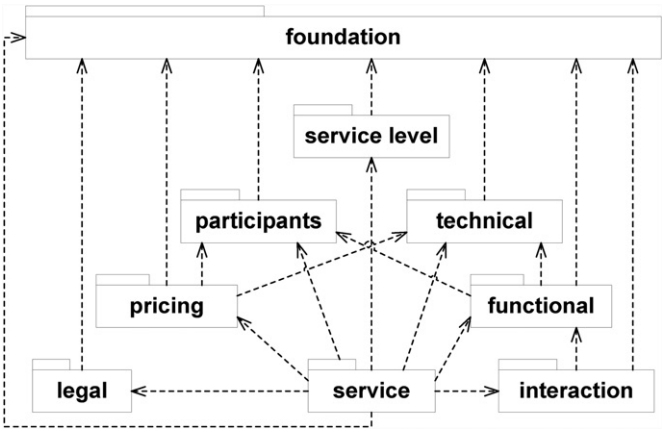

Fig. 5. UML package diagram of USDL (dependencies between packages are represented as arrows).

5.1. Service Module

The Service Module (depicted in Fig. 6) focuses on the essential *structure* of a service, i.e., the building blocks of a service, because of the complexity entailed. This arises from the conceptual diversity of service types, across the human-to-automation continuum (cf. Section 3.1), which brings services of different "shapes and sizes" into view. Also related to the structure of services are the orthogonal and adjacent issues of service *dependency*, related to the resources/services supporting a service's delivery, and composition, concerning the different parts put together for a service's functionality (cf. Section 3.3).

In the Service Module, services encapsulate functionality, from prior instrumental artifacts on a business or technical level. A **Service** can be manual, semi-automated and fully automated, or abstract, providing the template for concrete realization in a particular setting. One or more **ServiceVariants** allow access to predefined subsets of services. The packaging concept of services, **ServiceBundle,** allows a number of services (including **AbstractServices** and **ServiceBundles**) to be grouped without any execution relationship. A **CompositeService**, as a specialization of **Service,** is similar, except that the different parts of functionality, from potentially different services (including **AbstractServices**, **CompositeServices** and **ServiceBundles**) necessarily have an execution relationship. This can be an ordering of steps (like business processes), unordered steps (like program import) or data dependency (like widget mash-ups). Composition should not duplicate language-specific composition artifacts (like BPEL processes), but should apply when services, at the USDL level, have been composed. Since a major structural application of services is provisioning, **NetworkProvisionedEntity** generalizes both **Service** and **ServiceBundle** to this end, and associated descriptive classifications (e.g., terms and keywords), instrumental artifacts and shared variables for execution. Importantly, a **NetworkProvisionedEntity** has dependencies for concrete **Services**, **AbstractServices**, **ServiceBundles** and **Resources**, through the **Dependency** element. These relate to service delivery and reflect different dependency forms: ***Requires***, ***Includes***, ***Enhances***, ***Mirrors***, ***CanSubsitute*** and ***CanConflict***.

In order to illustrate the use of the Service Module, we introduce here a scenario from the domain of logistics. The scenario defines a number of example services that are provided by different organizations (see list of services in Table 1). Central to the scenario are the services of provider *P*. Provider *P* can be categorized as an advanced third-party logistics provider (3PL), which offers freight forwarding and transportation services, as well as IT-based services for order management and track and trace.

The first example is a *Road Transport* service that is provided through manual labor and corresponds to commodity "regional or national trucking services" (78101802) as defined by the United Nations Standard Product and Services Classification (UNSPSC). Another manual service is *Export Consolidation*, i.e., the process of containerizing cargo in a way that is optimal for its transport. In contrast, service *Shipment Manager* is an automated service providing access to booking and order management functions via EDI and Web interfaces. It has an independent ISO 9126 certification that is proven by a digitally signed PDF copy of the certificate. Fourthly, a more comprehensive service, *Ocean Export*, provides to shippers the entire capability of exporting cargo via sea shipment. It is a composite that consists of manual and automated steps and combines freight forwarding (UNSPSC commodity 78141501) and customs brokerage (UNSPSC 78141502) activities with other in-house assets, e.g., the provider's own trucking fleet (service *Road Transport*).

In addition to classic 3PL services, provider *P* offers enhanced tracking capabilities. For example, its service *Shipment Tracking* can be used to programmatically query and update the status of shipments. A service for *Subscription Management* allows for the subscription to particular status or events, of which the subscriber will be notified automatically. This includes the ability to define rules when and where to send such notifications. Users may register and subscribe to multiple event sources, such as *Shipment Tracking* of provider *P*. For convenience reasons, provider *P* offers a standard package *Tracking Event Management*, which is composed of *Shipment Tracking* integrated as a data source into *Subscription Management*. Because the integration between the two component

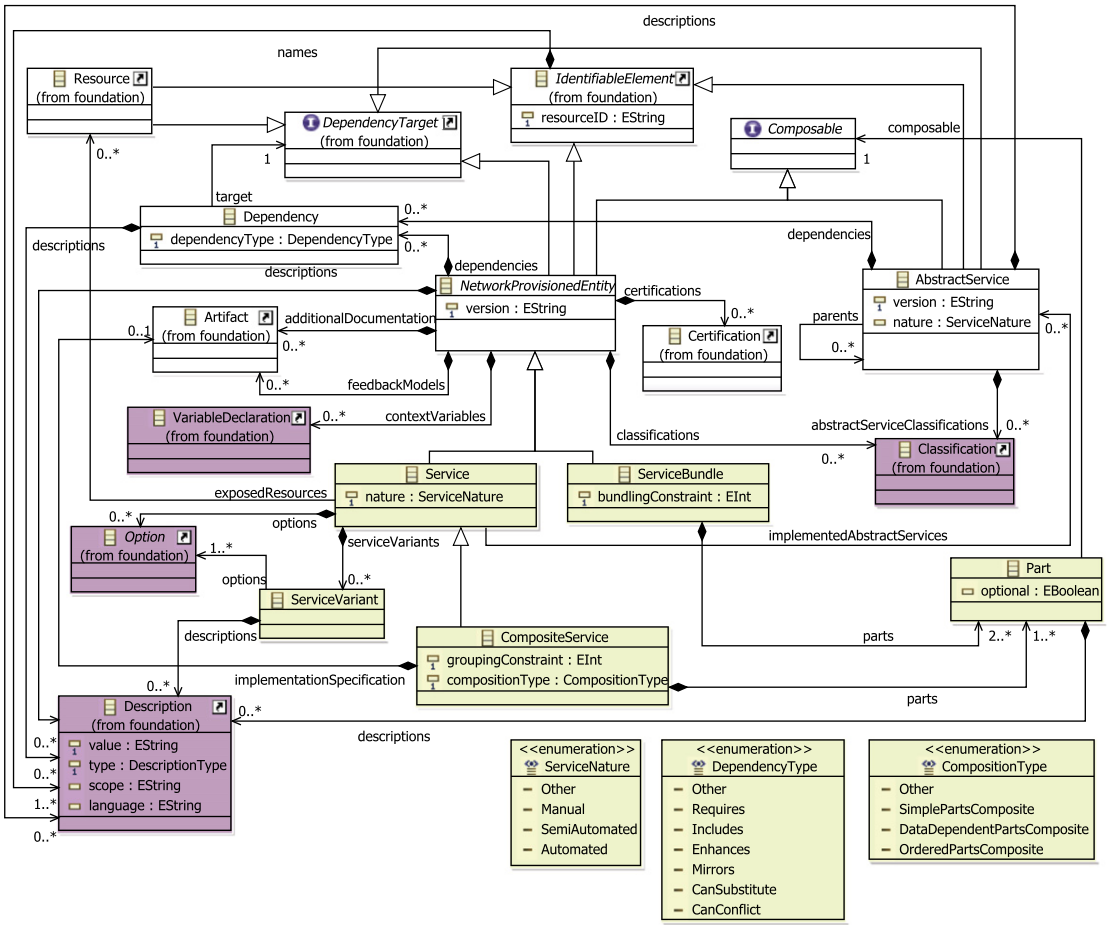

Fig. 6. Class diagram of the Service Module.

services (parts) is data-driven, it is classified as a **DataDependentPartsComposite** in USDL.

Other providers and services of the example scenario include the following.

- Provider X is a transport company that, among others, offers a *Track and Trace* service. The service is suitable as a data source for *Subscription Management* of *provider P*.
- Provider G runs a repository of world-wide digital maps that can be accessed through the Internet in different forms of rendering. The enterprise version of this service is available as *Maps* (*for commercial use*).
- Provider S is a specialist in traffic analysis and optimized route planning. Its core product is provided as a configurable, automated service called *Route Planning and Analysis*. In order to reach a wider audience among companies involved with domestic transport, provider S developed an application (and API) for *Traffic Monitoring*. It uses the service *Maps* of provider G to visualize user input, as well as data obtained from *Route Planning and Analysis*. The user can then request the calculation of an optimal route according to a number of specified parameters. *Traffic Monitoring* is realized as a process-driven orchestration using BPEL, which classifies as composition type **OrderedPartsComposite** in USDL. The process definition file (implementation specification) is attached to the USDL service description and has to be executed by the service consumer.
- A bundle *Adaptive Tracking* (*Road*) is created as a result of demand sensing on the service marketplace where all services are registered. It contains *Tracking Event Management*, *Shipment Tracking* and *Traffic Monitoring*, and should make it easy for shippers that do a lot of domestic freight to access route planning and track and trace functions, especially those of provider *P*. The bundle is offered by the marketplace operator with all three parties sharing discounts.

As mentioned earlier, the services in a service network or ecosystem manifest dependencies among each other and to resources. This is one of the important aspects of service delivery, because these dependencies define constraints on the usage and combination of services. In the service landscape of the example scenario there are several such dependencies between services, as depicted on the left side in Fig. 7 and listed in Table 2. Many of the dependencies are the result of acts of composition and bundling, e.g., service *Ocean Export* of provider *P* requires

Table 1
Service examples with properties covered by Service Module.

Provider	Service name	Type	Parts	Nature	Other
P	Road Transport	Service	–	Manual	Classification: UNSPSC, commodity 78101802
	Export Consolidation	Service	–	Manual	Classification: UNSPSC, commodity 78121501
	Shipment Manager	Service		Automated	Certification: ISO 9126 certified, copy of certificate at *http://…*
	Ocean Export	CompositeService	Road Transport, Export Consolidation	SemiAutomated	Composition type: DataDependentPartsComposite Classification: UNSPSC, commodity 78141501 & ~02
	Shipment Tracking	Service	–	Automated	Certification: ISO 9126 certified, copy of certificate at *http://…*
	Subscription Management	Service	–	Automated	Certification: ISO 9126 certified, copy of certificate at *http://…*
	Tracking Event Management	CompositeService	Subscription Management, Shipment Tracking	Automated	Composition type: DataDependentPartsComposite
X	Track and Trace	Service	–	Automated	–
G	Maps (for commercial use)	Service	–	Automated	–
S	Traffic Monitoring	CompositeService	Route Planning and Analysis, Maps	Automated	Composition type: OrderedPartsComposite implementation specification: *traffic_mon_203.bpel*
	Route Planning and Analysis	Service	–	Automated	–
- (service market place)	Adaptive Tracking (Road)	ServiceBundle	Traffic Monitoring, Tracking Event Management, Shipment Tracking	–	–

Road Transport and *Export Consolidation*. These **Requires** dependencies capture the fact that ordering *Ocean Export* necessitates ordering and paying for *Road Transport* and *Export Consolidation*. Defining a composite service in such a way allows for the easy setup of hierarchical service levels and pricing, because the service levels and pricing components for *Road Transport* and *Export Consolidation* do not have to be redefined in the service description of *Ocean Export*. They are automatically included in the overall package that constitutes the composite service.

Composites and their parts may also be completely unconnected, e.g., in the case of optional parts, or connected by dependencies of type **Includes**. For example, service *Tracking Event Management* of provider *P* requires *Shipment Tracking*, but includes *Subscription Management*. This means that, in order to use *Tracking Event Management*, consumers need a license for *Shipment Tracking*, while they get *Subscription Management* as part of the composite. Such **Includes** dependencies are the default for mandatory parts of a service bundle (see bundle *Adaptive Tracking*).

While the process of composition and bundling often gives rise to dependencies, the process is also itself influenced by dependencies. *Adaptive Tracking* (*Road*) is one example that shows how dependencies manifest themselves during composition and bundling (as represented by the arrows between the left and right side in Fig. 7). The bundle was created after the service marketplace sensed that there was a lot of demand for the combination of *Subscription Management* (resp. *Tracking Event Management*) and *Traffic Monitoring* in a certain user group. Because provider *P* wanted to push own services, the bundle had to include track and trace capabilities of provider *P*, which is why service *Tracking Event Management* was chosen instead of *Subscription Management*. At bundle creation, the bundling tool automatically made the suggestion to include *Shipment Tracking* due to the **Requires** constraint defined by *Tracking Event Management*.

Another form of dependencies influencing composition is illustrated by service *Track and Trace* of provider X. *Track and Trace* was registered on the service marketplace after provider *P* published its services there. Accordingly, the service marketplace tried to apply analytics and heuristics, in order to compute possible dependencies automatically. What it found is that *Track and Trace* has a similar interface to *Shipment Tracking* of provider *P*. It is therefore likely that *Track and Trace* provides the same functionality, albeit in a different setting (not tracking shipments of provider *P*), and could also be used together with *Subscription Management* in a composition similar to *Tracking Event Management*. Based on these results, the marketplace suggests a **Mirrors** dependency to *Shipment Tracking* and an **Enhances** dependency to *Tracking Event Management*. Both could lead a composition tool to suggest that *Track and Trace* is used as an additional data source by *Subscription Management*, e.g., as an optional part in a slightly different version of *Tracking Event Management* (Variant B in Fig. 7).

Dependencies are not necessarily always suggested by a system or tool, but can be defined manually. This is the case, for example, with service *Shipment Manager*.

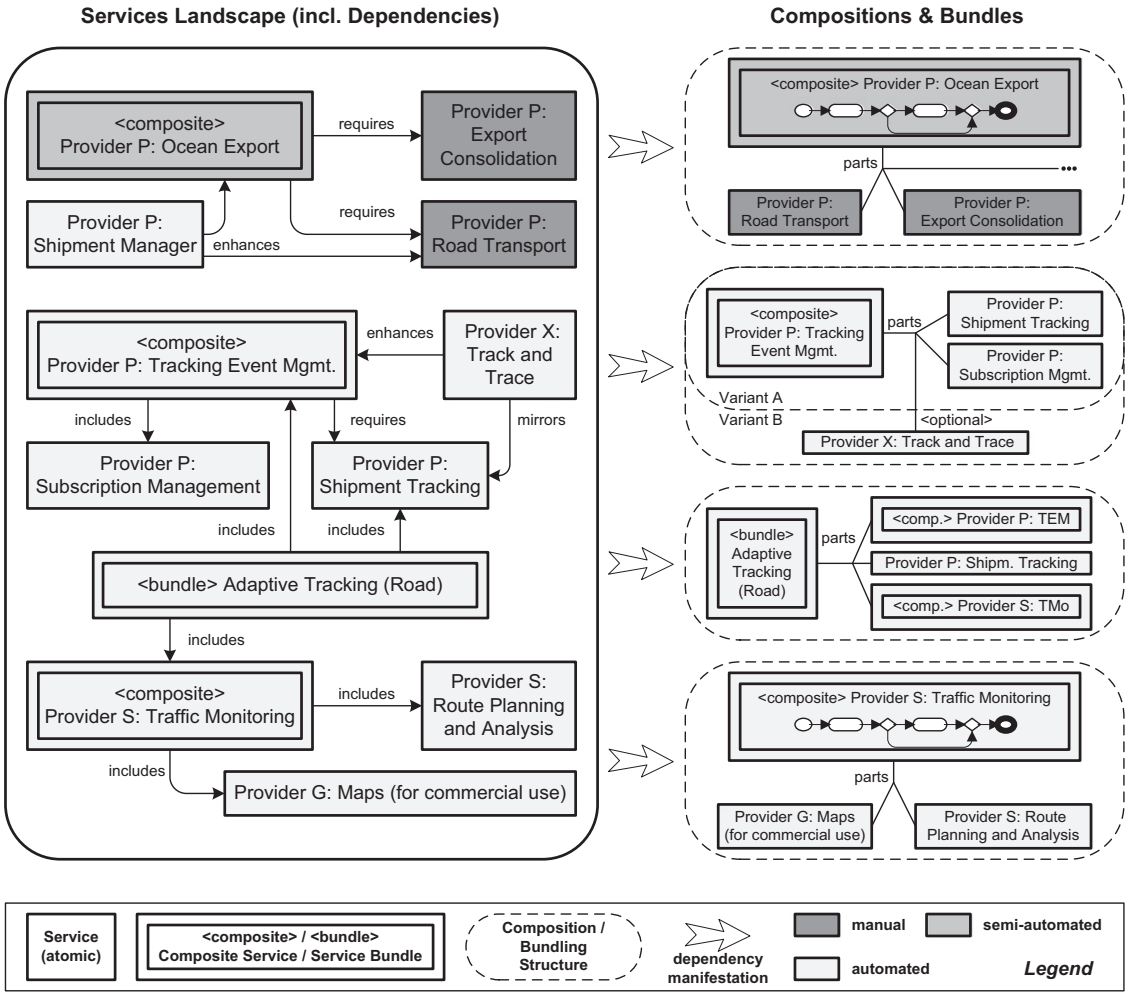

Fig. 7. Example services landscape with dependencies and individual composition structures.

Table 2
Dependencies among example services and to resources.

Service name	Service dependencies	Resource dependencies
Road Transport	–	–
Export Consolidation	–	–
Ocean Export	Requires: Road Transport, Export Consolidation	–
Shipment Manager	Enhances: Ocean Export, Road Transport	Requires: HTML3-compliant Web browser
Shipment Tracking	–	–
Subscription Management	–	Requires: OSGI-compliant JEE6 application server, JMS1.1-compliant message bus
Tracking Event Management	Requires: Shipment Tracking Includes: Subscription Management	Requires: OSGI-compliant JEE6 application server, JMS1.1-compliant message bus
Track and Trace	Mirrors: Shipment Tracking Enhances: Tracking Event Management	–
Traffic Monitoring	Includes: Route Planning and Analysis, Maps (for commercial use)	Requires: JBoss AS 5, jBPM 5.1
Route Planning and Analysis	–	Requires: 4 CPU kernels 2.4 GHz, 16 GB RAM, 4 TB storage, RedHat Linux
Maps (for commercial use)	–	–
Adaptive Tracking (Road)	Includes: Shipment Tracking, Tracking Event Management, Traffic Monitoring	–

Provider *P* knows that the service provides a direct, electronic channel for customers to book or query services, such as *Road Transport* or *Ocean Export*, thereby "enhancing" the possibilities to interact with these services.

Besides dependencies among services, there are also relationships where services depend on resources. As listed in Table 2, resource dependencies are most commonly requirements of automated services in regards to computing infrastructure. Such dependencies may range from very few to numerous, simple to complex, and abstract to concrete. To access service *Shipment Manager*, for instance, a user needs a Web browser that supports HTML v3, which is a single, more or less concrete resource that is widely available. The resource dependencies of *Tracking Event Management* are on an equally abstract level. Note that provider *P* does not operate *Subscription Management* or *Tracking Event Management*, because of limitations of its own infrastructure. When customers buy one of the automated services, they get the complete software application package, which they can deploy to a platform of their choice, subject to constraints expressed as resource dependencies. For example, in order to deploy and run the application package that enacts service *Tracking Event Management*, an OSGI-compliant JEE6 application server with message bus conforming to JMS 1.1 is required. This could be a combination of products from different vendors, e.g., JBoss AS 7 with JBoss SOA Platform 5 or IBM WebSphere AS 8 with WebSphere ESB. These products may run locally in the customer's own IT landscape, or be contracted as on-demand computing resources in the cloud.

In contrast to abstract resource dependencies, service *Traffic Monitoring* defines a set of very concrete requirements. It needs JBoss AS 5 with jBPM 5.1, mainly because provider S used features specific to this environment as part of the implementation. Finally, for service *Route Planning and Analysis* there is an option for customers to license the software and run it themselves. In this case, they need to provide an environment that meets the minimum requirements of 4 CPU kernels (e.g., 2 dual-core processors), each with a speed of 2.4 GHz, 16 GB main memory, 4 TB storage space and a RedHat Linux operating system.

This finishes the introduction of the example scenario for demonstrating the core elements of USDL and their application to a diverse range of services. The main purpose of these core elements, captured in the Service Module, is to (a) establish the unique entity that is the service (via naming and identification), and (b) position this entity in the service ecosystem that surrounds it. The latter includes categorizations and certifications associated with the service, as well as relationships to other services and resources (dependencies and composition) which govern its overall delivery and consumption.

5.2. Participants Module

The Participants Module captures the organizational actors that are important for the provisioning, delivery and consumption of a service. Since services captured in USDL should not expose or repeat the details of their instrumental artifacts, the details of organizational resources as stored in, e.g., organizational directories, are not central to USDL's concerns. However, sufficient details should be exposed in USDL so that a correspondence can be made to organizational actors where they directly concern service provisioning, delivery and consumption. The Participant Module connects **NetworkProvisionedEntity** with the participants in provisioning, delivery and consumption **Roles**, resonating the wider setting of a service ecosystem. As per the service discourse, the participant **Role** covers: service *owners* (cost center owners typically having governance responsibility of services); service *providers* (having service delivery responsibility); *stakeholders* (having regulatory, commercial or other designated interests in the service); *intermediaries* (having specialist provisioning, such as a broker or cloud provider, beyond the original provisioning); and end *consumer* (cf. Section 3.2 section). Basic organizational details are captured for these roles.

5.3. Functional Module

The Functional Module (depicted in Fig. 8) allows the capture of service functionality at an abstract level, regardless of the proximity of the service on the human-to-automation continuum, and free from technical implementation details. Conventionally, service functionality is captured as a set of operations. In contrast, USDL supports the capture of service functionality in different layers, for different levels of concern (white-box, grey-box and black-box view as discussed in Section 3.1). Accordingly, the Functional Module combines well-established concepts of capability modeling and functional decomposition. The element that abstracts service capabilities is **Function**, which is understood as representing a course of action. A **Function** may feature one or more input and output **Parameters**, as well as one or more **Faults** (related to exceptions). A **Function** has pre-conditions and produces post-conditions (effects). Two types of resources are defined for a **Function**, namely those used for performing (**utilizedResources**), e.g., tools or organizational roles, and those manipulated (**affectedResources**), e.g., business objects. The decomposition of **Function** into sub-functions supports different degrees of detail for different concerns of providers, intermediaries and consumers.

Table 3 shows functional properties of the services introduced previously. Service *Road Transport*, for example, provides point-to-point transportation on road according to the shipper's instructions utilizing 5-, 10- and 40-ton trucks, which means a single cargo item cannot exceed 30 t (40 t maximum gross weight minus 10 t vehicle weight).

5.4. Interaction Module

The Interaction Module (depicted in Fig. 9) captures the behavioral aspect of services and complements the structural focus of the Functional and Technical modules. The behavioral aspects of services concern how involved participants interact with the service. A large number of description languages have been proposed for modeling the external behavior of services as interaction protocols, and standards are now in wide used. As with the Technical Module, the design choice is not to replace existing efforts but to leverage them through the richer semantic

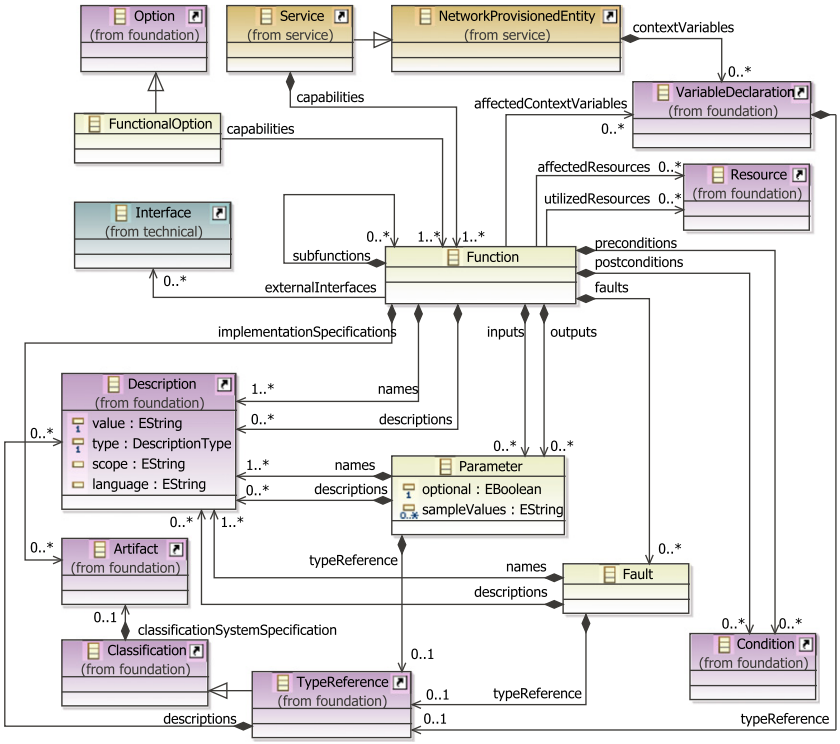

Fig. 8. Class diagram of the Functional Module.

Table 3
Service examples with properties covered by Functional Module.

Service name	Functions	Input/outputs	Conditions	Resources
Road Transport	Point-to-point transport on road	I: Shipping instructions, Cargo O: Cargo	Pre: weight of a single cargo item less than 30 t	Utilizes 5, 10 and 40 t trucks
Shipment Manager	Lookup rate	I: Shipment details O: Rate information		
	Order shipment	I: Shipment details O: Order confirm.	(*see Road Transport*)	
Ocean Export	Organize and manage export, Subfunctions: plan & book, orchestrate pre-carriage, process documents, orchestrate export	I: Invoice, shipping instructions, letter of credit, etc.	–	–

setting of USDL. The central concept in the module is **InteractionProtocol**, which groups the set of mandatory and optional interactions taking place between the participants. The **SimpleInteractionProtocol** is used to define a single sequence of interactions. An **Interaction** models an act of communication between the consumer of the service and one or more other participants that have a responsibility in delivery. These parties must be **Roles** of the Participants Module defined in the context of the service (feature **involvedRoles**). For more complex and long-running services, interactions can be grouped into phases of service delivery. A **ComplexInteractionProtocol** has a set of phases. A **Phase** holds a sequence of **Interactions** and requires as pre-conditions, and yields as post-conditions, a set of **Milestones**. A **Milestone**, in turn, is defined as the (formal or informal) description of state of objects that are affected by the service. It thus describes achievements, and phases may require certain milestones to be reached before they can start (feature preconditions). The interaction protocol of USDL maps straightforwardly to those in languages and techniques dedicated to service interaction behaviour. Language-specific models, such as the actual implementation specification, can be linked with **InteractionProtocol** through **Artifact**.

Table 4 illustrates examplary **InteractionProtocols** for two of the logistics services. For the sake of brevity, only one protocol is explained here. The *Road Transport* service follows a **SimpleInteractionProtocol** that is not supported through IT. Being a simple protocol it has no phases, only interactions. In order to invoke the service, a shipper may either request a quote and then place a

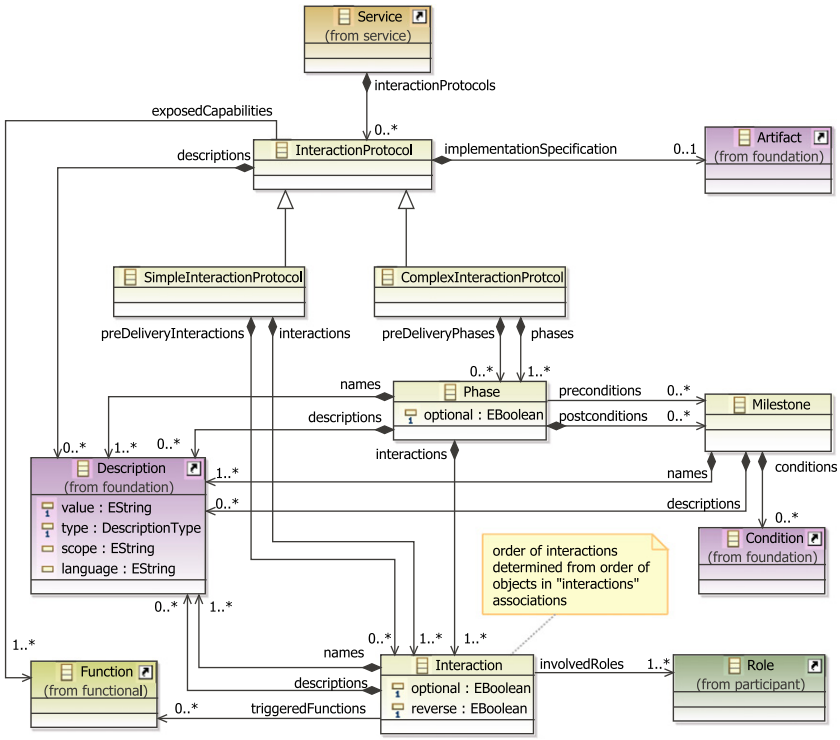

Fig. 9. Class diagram of the Interaction Module.

Table 4
Service example with properties covered by Interaction Module.

Service name	Protocol type	Phases	Interactions (r—reverse, o—optional)
Road Transport	Simple	–	Pre-delivery: Request quote (o), do booking (Place order) Delivery: pickup cargo at origin, drop off and hand over cargo at destination (r)
Ocean Export	Complex	Pre-delivery: order entry Delivery: booking and pre-carriage Delivery: export and main carriage	Place order, receive order confirmation (r, o) Provide documents and shipping instructions, receive booking confirmation (r), update instructions (o) Receive cargo receipt (r), receive change notification (r), receive draft waybill/bill of lading (r), receive final waybill/bill of lading and other documents (r)

booking, or immediately place a booking, e.g., if a contract with fixed rates exists. During delivery the cargo is picked up at the designated origin and handed over to the consignee at the destination.

5.5. Technical Module

The Technical Module supports a common way of describing the technical interfaces (access mechanism) of services. Following Semantic Web Services, the Technical Module serves the semantic association between technical interface descriptions and elements of USDL. The Technical Module is based on two significant styles used in practice, namely, operation-based and resource-based interfaces. Two corresponding interface classes mostly perform the function of a container for their elements (**OperationBasedInterfaceElement** and **ResourceBasedInterfaceElement**). These objects reference a particular element in an external interface description, e.g., an operation or parameter definition. A link to the actual interface description artifact (e.g., a WSDL file) that contains such definitions is provided through **implementationSpecifications.** A link is also provided to the USDL object implementing the interface **FunctionalElementRef.**

5.6. Pricing Module

The Pricing Module (depicted in Fig. 10) concerns the charging of services as mutually understood by those who

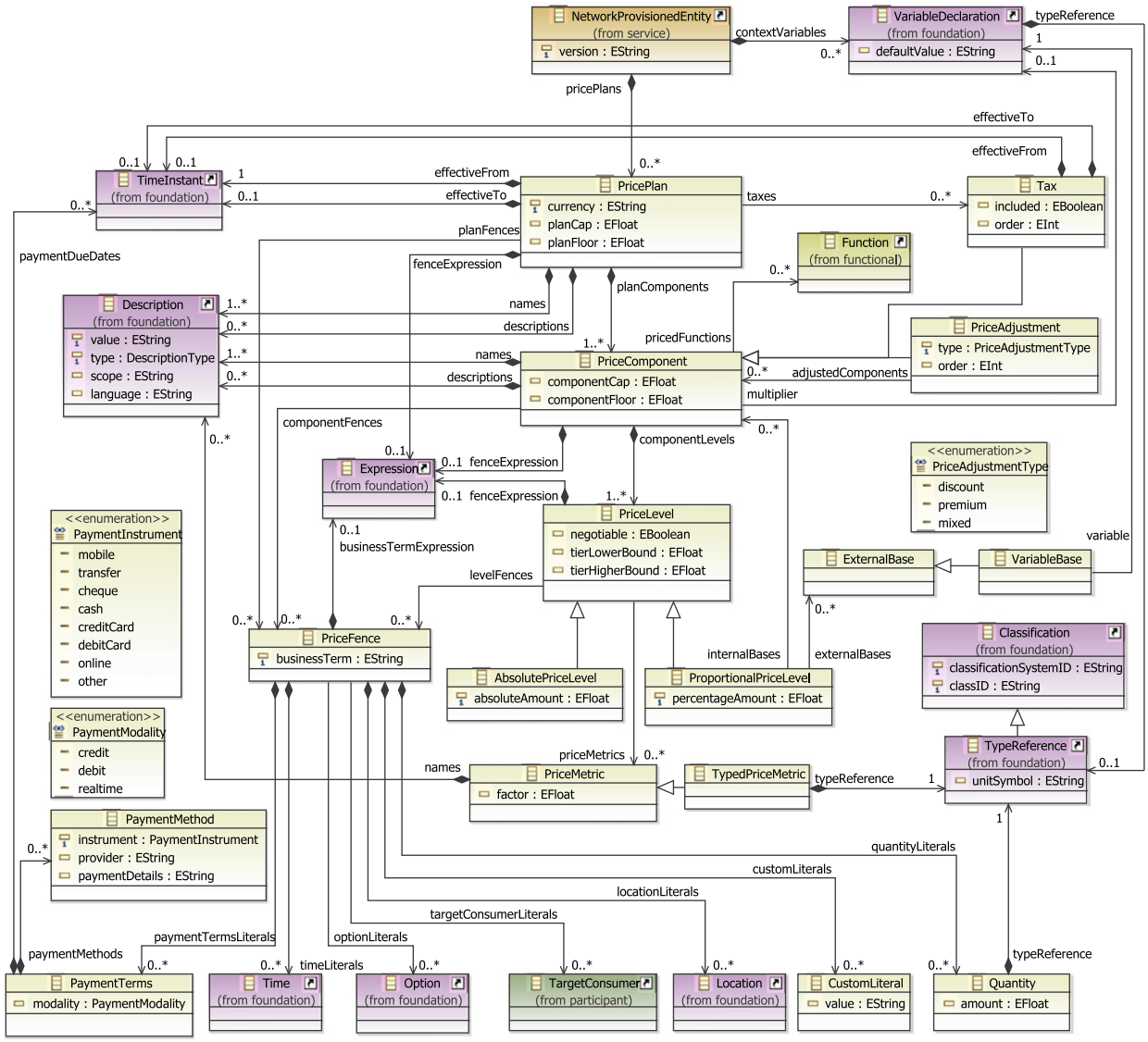

Fig. 10. Class diagram of the Pricing Module.

own or deliver services and those who consume them. For USDL, all segmented-pricing practices from the literature [36] and those that were considered relevant for USDL were assessed. USDL has a hierarchical structure for service pricing starting with one or more **PricePlans**. A **PricePlan** comprises one or more **PriceComponents**, related to different service capabilities (i.e., **Functions**), concerning different aspects of pricing, e.g., handset charge, domestic calls, or overseas calls for a telco service. Each **PriceComponent** has monetary values specified through **PriceLevel** which may be fixed per measurement (**AbsolutePriceLevels**) or proportional to a certain base (**RelativePriceLevels**). A **PriceLevel** has a fundamental **PriceMetric** as a measure upon which pricing is based. Discounts, surcharges, taxes, etc. are supported through **PriceAdjustments**. Dynamic variations on pricing such as rewards status of customer, bundled deals and other accepted negotiations with the customer, are supported through **PriceFence**. Collectively, the Pricing Module supports the representation of virtually any price structure encountered in the service industry, from simple single-tariff ones to complex price schedules involving multiple price plans. Segmenting conditions, which dictate when certain price elements apply, are also part of the module's capabilities.

The price plan examples in Table 5 give a good indication of the power and flexibility of the Pricing Module. The service *Ocean Export*, for example, offers a price plan that consists of multiple dependent price components. The main parts are the *Base Component*, a flat fee of 350 Euros for each order, and the *Cargo Component*, which prices per weight and distance. For shipments below 1 t, the shipper has to pay 35 Euros per 25 kg and 100 km driven. Above 1 t cargo weight the price falls to 25 Euros. In order to guard itself against

Table 5
Service example with properties covered by Pricing Module.

Service name	Price plan	Price components	Price levels	Price fences
Road Transport	Mileage-weight-based plan, currency: EUR	Tax: sales tax, priced function: point-to-point transport on road	Proportional level, 0.05 (5%), internal base: Cargo Component	–
		Cargo Component, priced function: point-to-point transport on road, multiplier: weight, dist.	Absolute level, 35 EUR per 25 kg and 100 km	Cargo weight less than 1 t
			Absolute level, 25 EUR per 25 kg and 100 km	Cargo weight above 1 t
Ocean Export	Service charges, currency: EUR	Base Component, priced function: organize & manage export	Absolute level, 350 EUR per order	–
		Cargo Component, priced function: (*as above*), multiplier: weight, distance	Absolute level, 35 EUR per 25 kg per 100 km	Cargo weight less than 1 t
			Absolute level, 25 EUR per 25 kg per 100 km	Cargo weight above 1 t
		Adjustment: currency adjustment factor, type: premium, floor: 30 EUR	Proportional level, 0.1 (10%), Internal base: Cargo Comp., Adjusted components: Cargo Comp.	–
		Tax: sales tax, priced function: organize & manage export	Proportional level, 0.05 (5%), Internal base: Base Comp., Cargo Comp., Currency Adj.	–

volatility of currency exchange rates, a *Currency Adjustment Factor* of 10%, minimum 30 Euros, is applied to the *Cargo Component*. Finally, the local sales tax of 5% on the sum of the three components is added on top.

5.7. Service Level Module

The Service Level Module is kept generic and does not specify how concrete service levels on concrete aspects should be specified. Instead, its main purpose is twofold. First, it provides the glue between abstractly specified service level issues in other USDL concepts. For example, it specifies to which elements of a **Function** a certain service level shall apply and the related **Role**. Second, it allows for incorporation of arbitrary attribute and expression languages. A **ServiceLevel** captures a single service level objective related to an offered, negotiated or agreed service. It is either related to state (**GuaranteedState**) or action (**GuranteedAction**) and has a **ServiceLevelExpression** capturing required assertions. A single participant, **ObligatedParty** (Participant Module), is obligated to enforce the service level. Service levels for given services are realized via **ServiceLevelProfile**. A set of service level specifications are combined into one profile and are offered, negotiated, or agreed upon as a whole. Different profiles can be used to specify different options of how service levels may be specified and grouped (e.g., as gold, silver, bronze profile). A **ServiceLevelProfile** resembles the concept of a service level agreement Template as for example specified in WS-Agreement.

5.8. Legal Module

The Legal Module (depicted in Fig. 11) addresses the need for legal certainty and compliance in service networks and in trading services on marketplaces. Participants need to know about the terms of usage of a particular service, for example, liability, privacy, or copyright. As a response, two variants of a Legal Module cover the modeling of licenses according to German and United States jurisdictions.[8] A detailed discussion of both modules and their differences can be found in [42]. The following explanation focuses on the German variant.

The class **Work** is central to the Legal Module since it represents the subject matter which can be licensed, i.e., a **Service**, **CompositeService**, or the service output. **UsageRights** can be granted for a **Work** according to different **UsageTypes.** The latter defines a specific, well-defined, economic manner of how to use a **Work**, e.g., the right to distribute. In the Legal Module, pricing is considered as a **Reward** for the rights holder for allowing other entities the usage of his work. Therefore, the class **PricePlan** of the Pricing Module represents *one* possibility to describe a **Reward**. The class **Function** in the Functional Module describes details about the usage of an artifact. This determines the actual **UsageRight** and its **UsageTypes**. The class **Role** in the Participants Module is used to describe a licensor. This also means that the licensor *has* to be specified in the Participants Module: the licensor either is the **Provider** of the service or has to be listed as a **Stakeholder**.

A simple example of a software license is listed in Table 6. The provider of *Shipment Manager* grants licensees a non-exclusive right to reproduce the service. This reproduction takes the form of accessing a Web UI through an

[8] The issue of dealing with different variants is further discussed in Section 6.

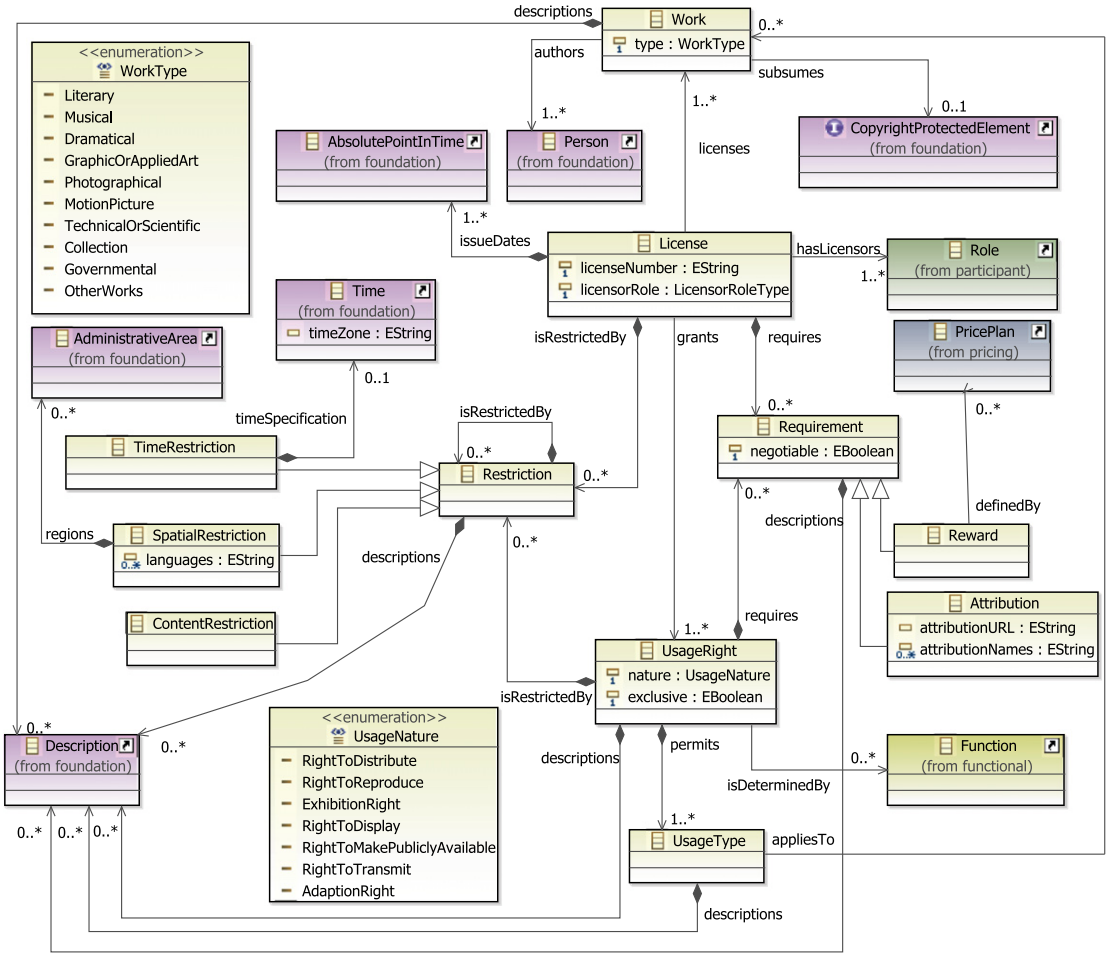

Fig. 11. Class diagram of the Legal Module (German variant).

Table 6
Service example with properties covered by Legal Module.

Service name	License	Work	Usage right
Shipment Manager	License to access via desktop client and Web UI, Licensor: provider, Licensor role: author	Subsumes: Shipment Manager, Author: John Doe (employed by provider)	Nature: RightToReproduce, exclusive: no, UsageType: use shipment manager client software on company PCs

Internet browser or installing a software client on PCs owned by the licensee. Even though the service and software is owned by the provider, German law requires a natural person to be identified as the author, which in this example is the developer of the software.

5.9. The Foundation Module

The Foundation Module (depicted in Fig. 13 in Appendix A) factorizes common parts of the remaining modules as a consistent continuation of modularization. Because of its basic character, all other modules depend on the Foundation Module meaning they reference one or more of its elements (e.g., general classes such as **Artifact**, **Description** or **NaturalPerson**).

6. How USDL supports requirements

With the overview of USDL modules, this section discusses how USDL addresses the requirements for service description languages aimed at the different types of services on the human-to-automation continuum.

Concerning the generic language requirements, USDL has clearly been positioned at the conceptual level, free of implementation details, given its UML-based meta-model and abstraction of business to technical aspects. In addition, it does not subsume related artifacts of services, such as technical WSDL interfaces, but rather points to them so that it serves as a unifying, but not an all-encompassing, language. This is a further indication of its adherence to *conceptualization*. USDL is defined by the *expressive power* of UML. The majority of the language definition is structural in nature, and, thus, has not made use of dynamic constraints. The *comprehensibility* of USDL is at the same level as UML meta-models used to describe many languages. With respect to *modularity*, the grouping of concepts has resonance with recognized sub-areas of services in existing service description languages, e.g., service, functional, interaction, pricing, service and legal, are treated as distinct conceptual areas in different languages as well as other unified service description proposals (e.g., [17]). Under current development, USDL lacks a *formal foundation*.

Concerning the service concept formation requirements, *organizational embedding* is inherent in USDL, seen through the various concepts directly corresponding to organizational concepts, e.g., (human or business level) services, resources and participants. Furthermore, a number of concepts has dependencies such that an explicit vertical alignment can be supported, from business to technical levels. This means that services across business and IT landscapes of an organization, and in business networks, can be captured through USDL and supported through services directories, repositories or registries. An organization's enterprise architecture tool that captures information, organizational and technical models could conveniently make reference to a USDL service repository, for instance. It could show how services are used in different parts of an organization and on different levels with other artifacts, such as business policies and capabilities, organizational actors, roles and resources, products, business processes and business objects. At the other end of the spectrum, a technical software registry could make reference to the USDL service repository for linking IT applications, platforms and infrastructure resources with both the technical and, ultimately, the business services that they support.

Regarding *cognitive sufficiency*, service functionality involves structural and behavioral aspects, with the complexity of details addressed through abstraction and decomposition. Service structure (*what*) is addressed through the Functional module, while service behavior is addressed by the Technical (*where*) and Interaction modules (*how*). Dedicated artifacts described in well established languages are referenced, not duplicated in USDL. Thus, a service provisioning tool could place descriptive attributes and dedicated models (e.g., based on BPMN, BPEL or SCA) side by side. Correspondences could be established at the tooling level such that highlighted attributes (e.g., **Functions**) could lead to highlighted parts of models. Stretching it further, non-functional properties of services, such as pricing, legal, SLA and security aspects, are also interlinked. For instance, a particular service offer's price plan could be viewed together with functional, interface and interaction details. Moreover, the details of price components and fences, as well as legal issues such as copyright clauses could be detailed down to more functions of a service. Taken together, next-generation editing tools for service provisioning and delivery could be considerably enhanced by bringing together service functionality with non-functional aspects. Moreover, consumer comprehension would be greatly enhanced by combining these aspects that are relevant to ordering, accessing and tracking non-trivial services. Fig. 12 provides a summary of conceptual correlations across USDL's different modules for further insights into USDL's cognitive sufficiency.

Although it concerns services and therefore the encapsulation of the internal functionality and reuse, there are wider considerations that USDL addresses concerning the *service information hiding* requirement. Service functionality, by way of functions, interfaces and interaction protocols, are abstractly described in USDL but concretely bound to external artifacts, such as WSDL and BPEL files. Such a two-tiered approach promotes a form of encapsulation and reuse. For instance, new functions can be added to services by reusing prior artifacts. Conversely, services within artifacts can reuse services described through USDL. For example, BPEL processes or SCA compositions can make use of services described through USDL. To promote reuse in large environments, artifacts themselves could be wrapped as USDL services so that they can be discovered and assessed fully for functional and non-functional compatibility with consuming services. Reuse then occurs on the level of USDL where the required functions of services could be composed with other service functions or be used as service dependencies. Therefore, USDL's support of service information hiding lends itself well to multi-facetted modeling of advanced service provisioning tooling envisaged above.

Given the positioning of USDL in a business network, the requirement of *deployment symmetry* is relevant. As explained in Section 4.2.4, it requires that services be operable regardless of where they are deployed including abiding by *a priori* organizational contexts set by providers of services. For example, roles, resources, services, and artifacts defined for a service should be available in a newly deployed environment where a service is being consumed. The question is what happens if these are similar but differently named in the new environment. A way to promote reuse in this situation is to provide abstract definitions of services (achieved by element **AbstractService**) so that these can be configured with concrete parts. A further support of the requirement is by allowing services to be conveniently extended at the level of a business network, e.g., extending a provider service for channeling.

The requirement for *execution resilience* relates to non-functional properties readily addressed in several modules. For example, the Pricing Module allows to represent "cooling off" periods by price fences should there be uncertainty about customer satisfaction concerning the fulfillment of a service. The Legal Module can capture licensing and copyright aspects as put in place by service

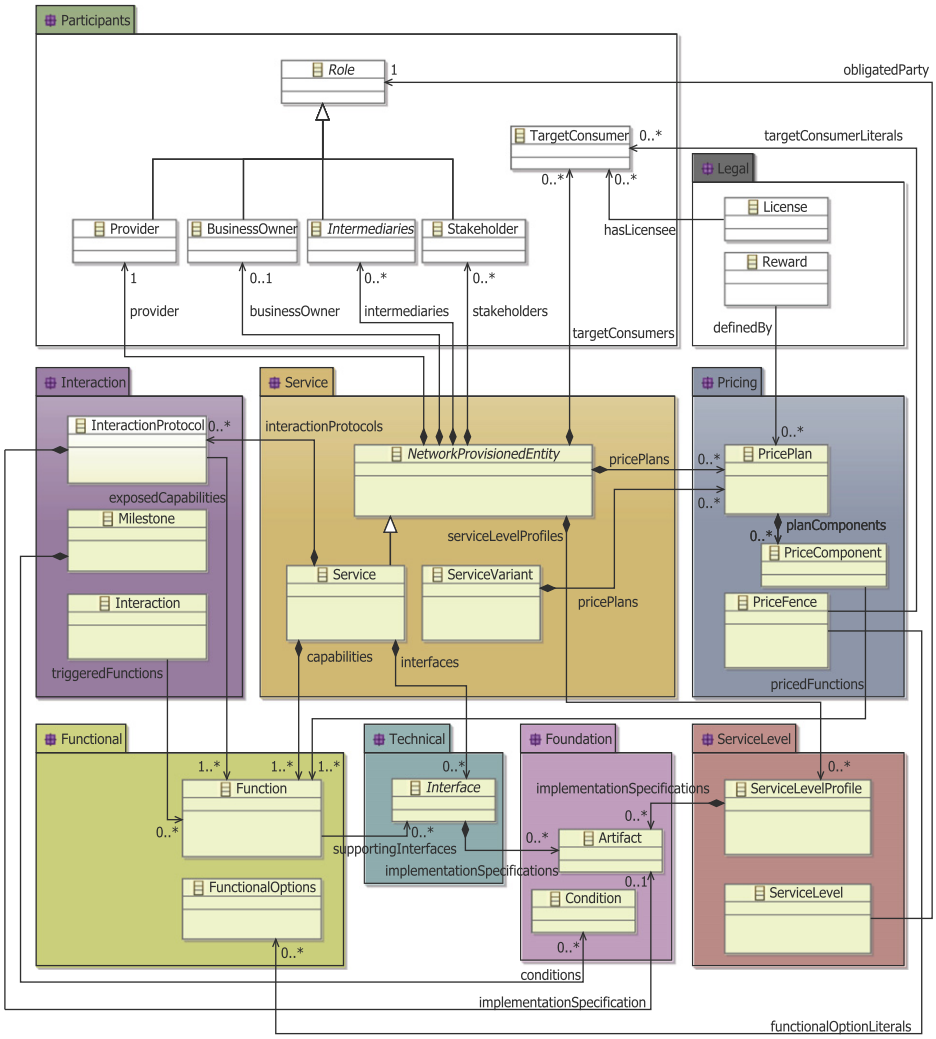

Fig. 12. Meeting the requirement of cognitive sufficiency leads to interconnected modules. Note that only interconnected elements of the individual modules are shown to give an impression. Also not apparent in the figure are "connecting interfaces" to reference targets that can be one of many USDL elements.

providers that will not be overridden when third parties extend and deliver services elsewhere. The Functional Module supports the definition of faults that can be mapped to technical fault messages (Technical Module) and exception handling procedures (Interaction Module).

7. Extensibility

Meeting the requirement of *extensibility* requires a more in-depth discussion following in this section. Different variants of USDL are required for different contexts, a fact that is already apparent in the Legal Module. Modeling of legal aspects such as **Licenses** depends on the legislation of the country and, consequently, different modules for the German and the United States common law have been developed. However, it is also apparent that there are overlaps between both modules. For instance, the class **Work** appears in both modules although with different attributes. In addition, there are also differences on the level of classes and relations. The class of **CopyrightTransfer** is apparent in United States but not in German law, the relation **authors** points to the foundational classes **Person** (German variant) and **Agent** (United States variant), respectively.

The approach in [41] presents one possible solution for managing variants of product catalogs which can be applied to USDL as well. Applying this solution requires representing USDL in a *canonical data model*, enriched by a *context driver mechanism*, *governance processes* and *appropriate tooling*. These four pillars will be described in the following.

Even if the terminology of the two Legal Modules follows a strict grammar, it is unlikely that separately introduced variants of USDL apply the same grammar and arrive at compatible naming schemes. Therefore, a *Canonical Data Model* (*CDM*) based on a standard naming convention ensures a common vocabulary. The Core Components Technical

Specification (CCTS) by the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) [37] is such a grammar and provides the platform independent definition of data types for document exchange and, in addition to that, a concept on how to organize, i.e., aggregate and associate, data on a higher level to create business documents (e.g., a purchase order).

CCTS can be used to specify how to assemble USDL variants and their data types as well as a *context driver mechanism* to create context-based instantiations of these general USDL variants. For example, the aforementioned **Work** might feature different attributes and relations in different business contexts. CCTS defines business context as the formal description of a specific business circumstance as identified by the values of a set of context categories, allowing different business circumstances to be uniquely distinguished. A context category is a group of one or more unordered values used to express a characteristic of a business circumstance. Examples for context categories are industry, business role, or country. Each context category consists of a set of related context values. Each value of the context category is defined by a classification scheme which is expressed as code list or identifier scheme. Attributes and relations of a class can then be modeled context-based, e.g., one can specify that the **authors** relation points to **Person** only when the context category **Country** equals **Germany**. Thus, both the German and the United States version of **Work** are merely two variants of a more comprehensive canonical data model.

The canonical data model and context driver mechanism are the prerequisites for a common vocabulary and avoid the proliferation of incompatible USDL variants. However, with the modification and adding of USDL variants by users, the canonical data model might lose structure, consistency, conciseness, and even correctness. Therefore, it has to be normalized, harmonized, optimized, corrected, and consolidated by a schema engineer or by an automatic analyzer both with the backup of a governance team and corresponding *governance processes*. Current business document standards, such as OAGIS [44] or RosettaNet [43], are typically managed centrally by such a governance board consisting of several schema engineers. Users can inquire changes to the schema and the governance board eventually decides if and how the change will be reflected in future versions. This task is non-trivial and time-consuming since a multitude of users might request similar extensions and alterations. The processes for the inquiry and changes are more or less explicit depending on the standards body.

The CCTS-based naming conventions, the canonical data model and the context driver are just specifications which need to be realized by appropriate tooling. In addition, the governance processes are usually implicit and realized without any tool support. A community of experts, i.e., the governance board, analyzes, discusses, and treats problems within the canonical data model. They have to manually identify these problems by analyzing the canonical data model on their own. Therefore, [41] proposes *appropriate tooling* to support and automate the aforementioned key features.

8. Evaluation

The design rationale of USDL ends with rigorously demonstrating its utility, quality, and efficacy. According to [38], this can be demonstrated by applying the following evaluation techniques: (i) feature comparison, (ii) theoretical and conceptual investigation, and (iii) empirical evaluation. The following section briefly sketches the evaluation endeavors in categories (i) and (ii), and goes into more detail regarding (iii).

First, Sections 3 and 4 contain a *feature comparison* between USDL and the surveyed efforts. It has been pointed out how both address the service discourse and requirements, concluding that USDL is a progression compared to the state of the art.

Second, USDL's ability to specify automated (software) services has been substantiated by a *theoretical evaluation*. Based on an analysis of literature on software description requirements and related approaches, a theoretically grounded evaluation framework has been derived and used as a benchmark to evaluate the constructs of USDL. According to the evaluation framework, comprehensive descriptions of software services should cover commercial information, implemented business semantics, technical binding information, and service quality. This evaluation showed that USDL provides the most detailed approach to date to comprehensively describe software services, which nevertheless should be harmonized in some aspects.

Third, several *empirical evaluations* have been carried out. In particular, the key ideas of the services discourse and the requirements have been empirically evaluated by a *survey*. A Delphi study [39] with 20 organizations has been conducted to assess how the two main features of information models are met by USDL. These aspects are, first, the reduction feature of conceptual modeling, i.e., "are the right aspects of a service selected, respectively, omitted?," and second, the pragmatic feature, i.e., "is the users' intended use of USDL compliant with its addressed scope?" The observations headmost suggest a very broad support of the respondents with the provided list of requirements.

Finally, Table 7 provides a coverage analysis, i.e., it summarizes which constructs of USDL have been used in different *case studies* in diverse walks of life. The first case study, viz., real-world services in Dresden (RWSDD), Germany, is documented in [40]. It concerned small and medium-sized companies, such as event agencies, design studios, translation and correction of documents, consulting and delivery services. Further, a B2B case study in Germany introduced a marketplace where car insurance companies can discover and consume services, such as car rental or repair, for their clients. Services, such as parking permit or business formation, and the interaction protocols of such institutional services were the focus of an eGovernment study in Australia. In addition, a large Australian bank evaluated USDL, including services such as account opening and their infrastructure dependencies (e.g., to installed mainframes). A logistics study comprised information (e.g., tracking), administrative (e.g., freight forwarding or customs filling), and movement services (e.g., transport, consolidation) which were quite different in their representation needs, but generally featured

Table 7
Coverage analysis. Each case study comprised several services. Consequently, cells represent the superset of required modeling constructs.

USDL		Case study						
		RWSDD	Car insurance	eGovernment	Banking	Logistics	Mobility	Cloud
Service	Type [*Service*, *CompositeService*, *ServiceBundle*]/ Nature [*Automated*, *SemiAutomated*, *Human*]	All/Human	All/All	All/All	All/All	All/All	Service/ Automated	Service/Automated
	Classifications [*yes*, *no*]/Certifications [*yes*, *no*]/ Dependencies [*Requires*, *Includes*, *Enhances*, *Mirrors*, *CanSubstitute*, *CanConflict*]	No/Yes/-	Yes/Yes/-	Yes/Yes/–	Yes/No/Includes	Yes/Yes/ Requires	No/No/–	No/No/Requires
	Variants [*yes*, *no*]	No	No	No	No	No	Yes	No
Service level	Profiles [*single*, *multiple*]/ Levels [*single*, *multiple*]/ Responsible [*provider*, *consumer*, *mixed*]	Single/Single/-	Single/Single/-	Single/Single/ Mixed	Single/Single/ Provider	Single/Single/ Provider	-/-/-	Single/Single/-
	General attributes [*security*, *reliability*, *performance*]	–	–	Security, performance,	Performance, reliability	Performance, reliability	Security	Security, reliability
	Availability [*Spatial*, *Temporal*]	Both	Both	Both	Spatial	Spatial	Temporal	Spatial
Price	Plans [*single*, *multiple*]/ Components [*single*, *multiple*]/ Levels [*single*, *multiple*]	Single/Single/Single	Multiple/Multiple/ Multiple	Single/Both/Both	-/-/-	Single/Both/ Both	-/-/–	Multiple/Multiple/ Multiple
	LevelType [*absolute*, *proportional*, *mixed*]	Absolute	All	All	–	All	–	Absolute
	Adjustments [*yes*, *no*]/ Fences [*no*, *plan*, *component*, *level*]	No/Component	Yes/All	Yes/Component, level	/-	Yes/Level	-/-	No/Level
Participants	Provider [*person*, *organization*]	Both	Organization	Organization	Organization	Organization	Organization	Organization
	Other participants [*business owner*, *stakeholders*, *intermediary*]	Business owner	–	Stakeholders	–	Business owner	–	–
Technical	Interfaces [*single*, *multiple*]	Single	Single	Single	Single	Multiple	Single	Multiple
	Type [*operation-based*, *resource-based*, *mixed*]	Operation	Operation	Operation	Operation	Operation	Operation	Operation
Interaction	Protocols [*single*, *multiple*]	–	Single	Single	Single	Single	Single	Single
	Types [*simple*, *complex*, *mixed*]	–	Simple	Mixed	Mixed	Mixed	Simple	Simple
	Replicated [*yes*, *no*, *mixed*]	–	No	No	–	No	No	No
Functional	Capabilities [*single*, *multiple*]/ Subfunctions [*yes*, *no*]	Multiple/No	Multiple/No	Multiple/Yes	Multiple/Yes	Multiple/Yes	Single/No	Multiple/Yes
	IOs [*yes*, *no*, *mixed*]/Conditions [*yes*, *no*, *mixed*]/ Resources [*yes*, *no*, *mixed*]	No/No/Yes	Yes/Yes/No	Yes/Yes/No	Yes/Yes/Yes	Yes/All/Yes	-/-/–	Yes/No/No
Legal	Terms of use [*yes*, *no*]	No	Yes	Yes	Yes	Yes	No	Yes
	Copyright licensing [*no*, *RightToDistribute*, *RightTo*…]	No	No	No	No	RightTo-Reproduce	No	No
Missing elements		Product-related service price plans	Domain-specific extensions	–	–	–	–	–

geographic availability and diverse pricing. A mobility case study comprised services such as route planning. Several cloud services, such as amazon's EC2, SAP's NetWeaver on demand, or SAP's Business ByDesign e-Recruiting extension, have been investigated. Such services required the modeling of dependencies, e.g., to browsers or tools, in particular.

The coverage analysis allows to judge whether USDL meets the case studies' modeling needs and whether USDL offers unnecessary modeling constructs. In essence, most constructs are required at least once, but also some deficits are apparent as indicated by the last row in Table 7. The Service, Participants, Functional and Service Level Modules are applied in each case study since USDL requires the existence of their central constructs for each service. For instance, there must not be a **Service** without a **Function**. Some case studies applied USDL for intra-organizational service cataloguing only, so the modeling of price plans was not always required. Note, however, that intra-organizational service cataloguing is a prerequisite for exposing services to external business networks. According to our experiences with the case studies, service cataloguing therefore happens as a first step and exposure is planned as the actual goal of the exercise. An interesting observation is the pervasive use of the Technical Module. This shows that Web services were apparent in each case study and the need to semantically associate technical interface descriptions with elements of USDL. The Legal Module was only recently finished what explains its low use so far. The case studies also underline that a wide range of artifacts, across economic, political, commercial, community and individual settings are referred to as services (cf. Section 3.1). Indeed, we have observed that most services in the case studies lie on a human-to-automation continuum.

Essentially, description efforts tend to increasingly abstract from implementation-specific artifacts such that their generation and maintenance becomes more effective. For example, companies invested in SOA because it promises significant productivity gains through abstracting functionality on the technical level. Accordingly, the accompanying SOA description efforts (in particular, the WS-* approaches) enable working on a standardized and more abstract level than proprietary and heterogeneous implementation-specific artifacts. USDL continues this trend and is the most comprehensive approach for abstracting also business-related functional and non-functional properties to the best of our knowledge.

Since enterprise applications affect many resources that relate to a variety of such implementation-specific artifacts, a USDL-based repository should not live in isolation in an IT architecture. In particular, services relate to business processes, applications, objects and resources. There are corresponding methods, languages, and tools for managing these (e.g., business process management, enterprise architecture, and model-driven architecture frameworks, IT governance portolio systems, etc.). Therefore, USDL has to be integrated, aligned with, and supported through such wider systems frameworks in order to coherently manage descriptions of these artifacts. This results in more cost effective development and maintenance because business requirements can be specified on the more abstract level of system specifications including service descriptions. Changes on this level can then be automatically propagated to associated implementation-specific artifacts. For example, a price plan can be maintained in an abstract, purely business-oriented way with USDL and then transformed to a more technical level and fed into billing engines for execution. SLAs specified in USDL can be input to a monitoring engine as discussed in [46] and so on.

9. Conclusion and outlook

The paper represents an important consolidation for describing business, operational and technical aspects of services. The consolidation required a coherent design rationale absorbing ideas and concepts across a variety of different strands, viz., SOA, Semantic Web Services, Business, Service Network, and Service System approaches. The design rationale of the resulting Unified Service Description Language (USDL) includes a survey of related approaches, the identification of a proper universe of discourse, language design requirements, and evaluation. By feature comparison, the evaluation shows that USDL supersedes the related approaches since they typically focus on specific aspects only.

A comprehensive USDL tool chain, including editors, stores and marketplaces, has been developed by SAP Research and partners, allowing flexible deployment scenarios.[9] Yet, more methods are required to fully exploit USDL, such as integration with service engineering approaches (e.g., SCA). Future research should also investigate graphical representations of the language in order to lower the entrance barrier. The fact that USDL involves multiple disciplines is both an advantage and poses a challenge. Background in all involved disciplines is required in order to comprehensively describe a service with USDL. Another endeavor, entitled *Linked USDL*, targets simplification and reuse of existing schemas on the basis of Linked Data principles and technologies [47]. The Linked USDL community has been established in order to foster an open development as well as a broad and world-wide adoption. Further, USDL should incorporate a formal foundation against which the intended semantics of USDL can be expressed. This can be achieved by drawing from the underlying logics of Semantic Web Services approaches, for instance.

Finally, every effort to describe services in a generic form faces the challenge of extensibility. Therefore, a successful mechanism for extensibility is of paramount importance if USDL is to be applied on a large scale. The approach for extensibility presented in this paper can be seen as a proposal but has not been tested in practical settings.

Role of the funding source

This work has been funded by the TEXO project within the THESEUS research program initiated by the German Federal Ministry of Economy and Technology, projects funded by the German Federal Ministry of Education and

[9] Specifications, XML schema, example files and tools can be freely obtained at http://www.internet-of-services.com.

Research projects (e.g., Premium Services), European Union DG INFSO projects (e.g., FAST, RESERVOIR, MASTER, ServFace, SHAPE, SLA@SOI, SOA4ALL), and the Australian Smart Services CRC.

Acknowledgements

USDL was developed across several locations, including SAP Research Belfast, Brisbane, Darmstadt, Dresden, Karlsruhe, and Sophia Antipolis. Apart from the authors, a multitude of researchers contributed to the language among them Christian Baumann, Michele Bezzi, Jorge Cardoso, Anis Charfi, Luca Compagna, Tom Kiemes, Norman May, Francesco Novelli, Philip Robinson, Benjamin Schmeling, Gabriel Serme, Wolfgang Theilmann, Matthias Winkler, and Heiko Witteborg. In addition, we would like to thank the Fraunhofer departments FOKUS and IAO as well as the Universities of Augsburg, Dresden, Münster, Germany, Smart Services CRC, Australia, as well as Anke Thede, Josef Spillner, for their contributions.

Appendix A

See Fig. 13.

References

[1] A. Hevner, S. March, J. Park, S. Ram, Design science in information systems research, MIS Quarterly 28 (1) (2004) 75–105.
[2] A.J. Berre. Service-oriented Architecture Modeling Language (SoaML)—Specification for the UML Profile and Metamodel for Services (UPMS). OMG Document. 〈http://www.omg.org/spec/SoaML/1.0/Beta2/〉, 2009.
[3] C. M. MacKenzie, K. Laskey, F. McCabe, P.F.Brown, R.Metz, Reference Model for Service Oriented Architecture 1.0, Official OASIS Standard, October 2006, 〈http://www.oasis-open.org/committees/soa-rm/〉, 2006.
[4] C. Harding, Service-oriented Architecture Ontology. The Open Group Draft 2.0, 〈http://www.opengroup.org/projects/soa-ontology/〉, 2008.
[5] B. Pandit, V. Popescu, V. Smith, SML Service Modeling Language. V1.1 W3C Recommendation, May 2009. 〈http://www.w3.org/TR/sml/〉, 2009.
[6] S.A. McIlraith, T.C. Son, H. Zeng, Semantic Web Services, IEEE Intelligent Systems 16 (2) (2001) 46–53.
[7] A. Ankolekar, M.H. Burstein, J.R. Hobbs, O. Lassila, D.L. Martin, S.A. McIlraith, S. Narayanan, M. Paolucci, T.R. Payne, K.P. Sycara, H. Zeng, DAML-S: Semantic markup for Web services, in: F. Isabel, Cruz Stefan, Decker Jérôme, Euzenat Deborah, L. McGuinness (Eds.), The Emerging Semantic Web, Selected Papers from the First Semantic Web Working Symposium, Stanford University, California, USA, July 30–August 1, 2001, Frontiers in Artificial Intelligence and Applications, IOS Press, 2001, p. 75.
[8] D. Roman, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, D. Fensel, WWW: WSMO, WSML, and WSMX in a nutshell, in: Proceedings of the Asian Semantic Web Conference (ASWC), LNCS 4185, Springer, 2006, pp. 516–522.
[9] J. Farrell, H. Lausen, Semantic Annotations for WSDL and XML Schema (SAWSDL), W3C Recommendation, August 2007, 〈http://www.w3.org/TR/sawsdl/〉, 2007.
[10] K. Gomadam, A. Ranabahu, A. Sheth, SA-REST: Semantic Annotation of Web Resources, W3C Member Submission, April 2010, 〈http://www.w3.org/Submission/SA-REST/〉, 2010.
[11] B. Norton, M. Kerrigan, A. Mocan, A. Carenini, E. Cimpian, M. Haines, J. Scicluna, M. Zaremba, Reference ontology for semantic service-oriented architectures. OASIS Public Review Draft 0.1, November 2008, 〈http://docs.oasis-open.org/semantic-ex/ro-soa/v1.0/pr01/see-rosoa-v1.0-pr01.html〉, 2008.
[12] C. Pedrinaci, J. Domingue, Towards the next wave of services: linked services for the Web of data, Journal of Universal Computer Science 16 (13) (2010) 1694–1719.
[13] D. Fensel, F. Fischer, J. Kopecký, R. Krummenacher, D. Lambert, T. Vitvar, WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web, W3C Member Submission, 23 August 2010, 〈http://www.w3.org/Submission/WSMO-Lite/〉, 2010.
[14] J. Kopecky, T. Vitvar, C. Pedrinaci, M. Maleshkova, RESTful services with lightweight machine-readable descriptions and semantic annotations, in: Wilde Erik, Pautasso Cesare (Eds.), REST: From Research to Practice, Springer, 2011.
[15] DIN PAS, German standard for the description of services for tendering, Beuth Verlag GmbH Ref. Nr. PAS 1018: 2002–12, 2002.
[16] A. Emmrich, Ein Beitrag zur systematischen Entwicklung produktorientierter Dienstleistungen, Dissertation, Wirtschaftsinformatik, vol. 170, HNI-Verlagsschriftenreihe, Paderborn, Heinz Nixdorf Institut, Universität Paderborn, 2005, ISBN 3-935433-79-4, 2005.
[17] J. O'Sullivan, Towards a Precise Understanding of Service Properties, Dissertation, Faculty of Information Technology, Queensland University of Technology, 2006, pp. 1–232.
[18] I. Toma, Modeling and ranking Semantic Web Services based on Non-Functional Properties, Ph.D. Thesis, Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck, 2010.
[19] Z. Baida, J. Gordijn, H. Akkermans, Service Ontology, Free University, Amsterdam, 2001.
[20] H. Akkermans, Z. Baida, J. Gordijn, N. Peña, A. Altuna, I. Laresgoiti, Value Webs: using ontologies to bundle real-world services, IEEE Intelligent Systems (EXPERT) 19 (4) (2004) 57–66.
[21] S. De Kinderen, J. Gordijn, e3Service—an ontological approach for deriving multi-supplier IT-service bundles from consumer needs, in: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, IEEE Press, 2008, pp. 318 (1–10).
[22] M. Bitsaki, O. Danylevych, W. van den Heuvel, G. Koutras, F. Leymann, M. Mancioppi, C. Nikolaou, M. Papazoglou, An architecture for managing the lifecycle of business goals for partners in a service network, in: Petri Mähönen, Klaus Pohl, Thierry Priol (Eds.): Towards a Service-Based Internet, First European Conference, ServiceWave 2008, Madrid, Spain, December 10–13, 2008. Proceedings. Lecture Notes in Computer Science 5377 Springer, 2008.
[23] S. Alter, Service system fundamentals: work system, value chain, and life cycle, IBM Systems Journal 47 (1) (2008) 71–85.
[24] R. Ferrario, N. Guarino, C. Janiesch, T. Kiemes, D. Oberle, F. Probst, Towards an ontological foundation of services science: The general service model, in: A. Bernstein, G. Schwabe (Eds.), 10th International Conference on Wirtschaftsinformatik, 16–18 February 2011, Zurich, Switzerland, vol. 2, 2011, pp. 675–684.
[25] D. Oberle, N. Bhatti, S. Brockmans, M. Niemann, C. Janiesch, Countering service information challenges in the Internet of Services, Journal of Business & Information System Engineering (BISE) 1 (5) (2009) 370–390.
[26] J. A. Estefan, K. Laskey, F.G. McCabe, D. Thornton, Reference Architecture Foundation for Service-Oriented Architecture, Version 1.0, Committee Draft 2, October 2009, OASIS Service-Oriented Architecture Reference Model TC, 2009.
[27] K. A. Dhanesha, A. Hartman, A. N. Jain, A model for designing generic services, in: Proceedings of the Seventh IEEE International Conference on Services Computing, IEEE Press, 2009, pp. 435–442.
[28] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, Second Revised edition, Addison-Wesley Longman, Amsterdam, 2008.
[29] A. Barros, U. Kylau, Service delivery framework—an architectural strategy for next generation service delivery in business network, in: P. Kellenberger (ed.), Proceedings 2011 Annual SRII Global Conference SRII 2011, 30 March–2 April 2011, San Jose, California, USA, IEEE Computer Society Conference Publishing Services (CPS), 2011, pp. 47–58.
[30] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, S. Dustdar, Moving applications to the cloud: an approach based on application model enrichment, International Journal of Cooperative Information Systems (IJCIS) 20 (2011) 3.
[31] M. Lankhorst, Enterprise Architecture at Work. Second ed, Springer, Berlin Heidelberg, The Enterprise Engineering Series, 2009.
[32] J. Griethuysen, Concepts and Terminology for the Conceptual Schema and the Information Base, Technical Report, ISO/TC97/SC5/WG3, 1982.
[33] A. Barros, A. ter Hofstede, Towards the construction of workflow-suitable conceptual modelling techniques, Information System Journal 8 (4) (1998) 313–337.

[34] E. Falkenberg, W. Hesse, P. Lindgreen, B. Nilsson, J. Oei, C. RollandFRISCO—A Framework of Information System Concepts—the FRISCO Report, Technical Report IFIP WG 8, 1 Task Group FRISCO, 1998.
[35] A. Barros, C. Baumann, A. Charfi, M. Flügge, S. Heinzl, T. Kiemes, U. Kylau, F. Marienfeld, N. May, O. Müller, F. Novelli, D. Oberle, J. Pattberg, P. Robinson, B. Schmeling, W. Theilmann, H. Witteborg, J. Finzen, A. Horch, M. Kintz, . Unified Service Description Language (USDL) Module specifications. Version 3.0, Milestone M5, SAP Research, May 2011. Available at 〈www.internet-of-services.-com〉, 2011.
[36] T.T. Nagle, J.E. Hogan, The Strategies and Tactics of Pricing, Fourth ed. Prentice Hall, 2005.
[37] G. Stuhec, G. Crawford, How to Solve the Business Standards Dilemma—The CCTS Standards Stack, SAP Developer Network (SDN) Article 2006, November 2006.
[38] K. Siau, M. Rossi, Evaluation techniques for systems analysis and design modelling methods—a review and comparative analysis, Information Systems Journal 21 (3) (2011) 249–268.
[39] M. Benarie, Delphi and Delphi like approaches with special regard to environmental standard setting, Technological Forecasting and Social Change 33 (2) (1988) 149–158.
[40] J. Spillner, R. Kursawe, A. Schill, Case study on extending Internet of Services Techniques to real-world services, in: Proceedings of the Second International Symposium on Services Science (ISSS'10), Leipziger Beiträge zur Wirtschaftinformatik, Bd. 6, Logos Verlag, Leipzig, Germany, 2010.
[41] Stuhec, G. Using CCTS Modeler Warp 10 to Customize Business Information Interfaces, SAP Developer Network (SDN) Article 2007, November 2007.
[42] C. Baumann, C. Loës, Formalizing copyright for the Internet of Services, in: G. Kotsis, D. Taniar, E. Pardede, I. Saleh, I. Khalil (Eds.): iiWAS'2010—The 12th International Conference on Information Integration and Web-based Applications and Services, 8–10 November 2010, Paris, France. ACM 2010, 2010, pp. 714–721.
[43] S. Damodaran, B2B integration over the Internet with XML: RosettaNet successes and challenges, in: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (WWW Alt. '04). ACM, New York, NY, USA, 2004, pp. 188–195.
[44] M. Flebowitz, OAGIS 8.0: practical integration meets XML schema, XML Journal 03 (09) (2002).
[45] A. Barros, D. Oberle, Handbook of Service Description: USDL and Its Methods, Springer, New York, 2012.
[46] J. Spillner, M. Winkler, S. Reichert, J. Cardoso, A. Schill, Distributed contracting and monitoring in the Internet of Services, in: T. Senivongse, R. Oliveira (Eds.): Distributed Applications and Interoperable Systems, nineth IFIP WG 6.1 International Conference, DAIS 2009, Lisbon, Portugal, June 9–11, 2009. Proceedings. Lecture Notes in Computer Science 5523, 2009, pp. 129–142.
[47] C. Bizer, The emerging web of linked data, IEEE Intelligent Systems 24 (5) (2009) 87–92.
[48] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, W3C Working Group Note 11, February 2004.

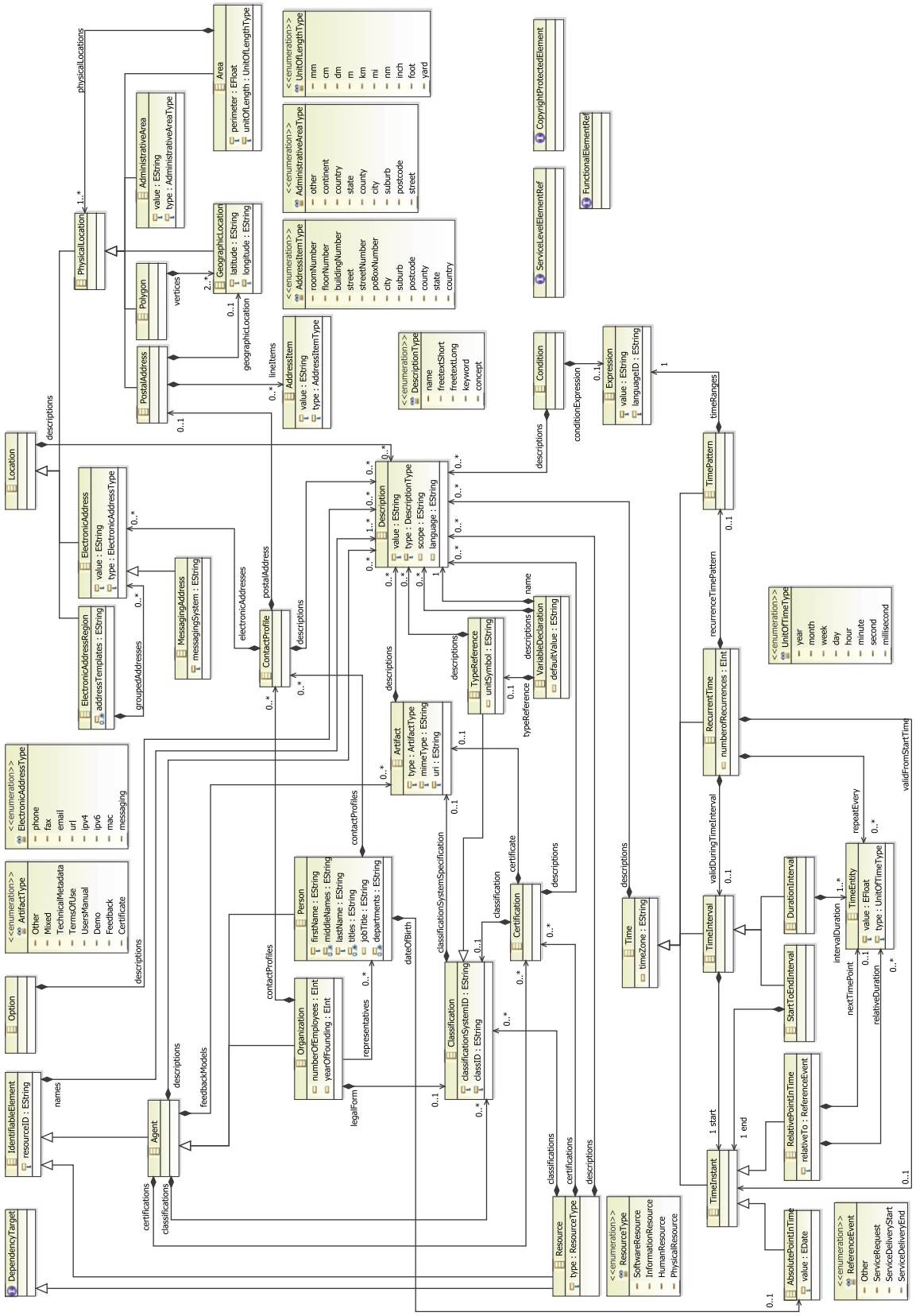

Fig. 13. Class diagram of the Foundation.