Technical Report:
An Ontology of Services and Service Descriptions

Aldo Gangemi
Laboratory for Applied Ontology,
Institute for Cognitive Sciences and Technology,
National Research Council, I-00137 Rome, Italy
gangemi@ip.rm.cnr.it

Peter Mika, Marta Sabou
Vrije Universiteit, Amsterdam
de Boolelaan 1081a, 1081HV Amsterdam
pmika, marta@cs.vu.nl

Daniel Oberle
Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
oberle@aifb.uni-karlsruhe.de

November 28, 2003

1 Introduction

This Technical Report covers original work by the authors on an Ontology of Services and Service Descriptions. This work has been initiated within the European WonderWeb project [won].

The WonderWeb architecture envisages a tight integration between web-based KR languages, ontology learning and manipulation tools, foundational ontologies and ontology building methodologies.

WonderWeb also provides an infrastructure that facilitates plug'n'play engineering of ontology-based modules and, thus, the development and maintenance of comprehensive Semantic Web applications, an infrastructure which is called *Application Server for the Semantic Web (ASSW)* [OVMS03]. It facilitates re-use of existing modules, e.g. ontology stores, editors, and inference engines, combines means to coordinate the information flow between such modules, to define dependencies, to broadcast events between different modules and to translate between

ontology-based data formats. Since software modules come as black boxes of code, descriptions need to be attached to them in order to facilitate their discovery. As a result, the ASSW features a registry that stores descriptions of the module and its API. Such descriptions adhere to an ontology which is not only used for module and API discovery, but also for manual classification, connectivity and implementation tasks. An Application Server for the Semantic Web can therefore be considered as *semantic middleware*. Additionally, there exists the possibility to offer a module's functionality by another paradigm. E.g., the module might not only be represented as one object revealing a particular API, but its functionality may also be accessible as separate web services. This is achieved by translating a module's ontological API description into corresponding web service descriptions.

Existing conceptualizations of web services, such as the Web Services Architecture (WSA) [BCF+03] are informal and thus cannot avoid ambiguities even in the very definition of web services (see Section 7). Ontologies for the descriptions of web services, in particular DAML-S [Coa03] and its successor OWL-S, attempt to cater for both worlds, but make no distinction as to what are general aspects of services and what are the notions specific to software or web services in particular. As a result, confusion arises as to the nature of objects comprising and processed by web services (see Section 6).

Therefore, the initiative was taken within the project to create an Ontology of Services using the DOLCE foundational ontology, which has been also developed within the project. The resulting "upper ontology" of services based on well-founded principles is expected to influence (support) the design of more specific ontologies, such as the one designed for the description of software modules in the ASSW use case. It was also confirmed that the Ontology of Services would help in clearing up otherwise fuzzy definitions of concepts related to web services and in pointing out inconsistencies or ambiguities in conceptualizations such as the WSA document.

The Ontology of Services is thus part of a layered architecture of ontologies developed within WonderWeb (cf. Figure 1). On the one hand, it is an extension (module) to the DOLCE foundational ontology [MBG+02]. In particular, it makes extensive use of the Ontology of Descriptions (also called Descriptions & Situations or D&S) available in the extended version of DOLCE, called DOLCE+ [GM] (see also Section 4). On the other hand, the Ontology of Services generalizes notions of existing conceptualizations of web services or web service descriptions such as the DAML-S [Coa03], the Web Services Architecture [BCF+03] or the Ontology of Software Modules used within the ASSW [OSRV03]. More specifically, the Ontology of Services covers all kinds of services, with information services as a special case. At the bottom layer of the architecture we find domain-level ontologies. An example of such an ontology is the ontology of Semantic Web tools, which provides descriptions directly processed by the ASSW.

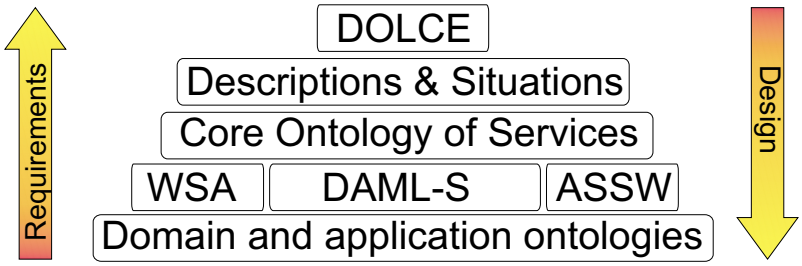

Figure 1: Ontology stacking in WonderWeb.

Our method was a combination of a bottom-up and a top-down approach. On the one hand, ontologies in the lower layers provided representation requirements for the higher layers, which abstracted their concepts and relationships. On the other hand, the upper layers provided design guidelines to the lower layers.

In the following, we will use the example of a typical (but hypothetical) web-based flight booking service to illustrate some of the notions introduced.

2 Motivation

We share the motivation of the DAML coalition that descriptions of (web) services should be formulated according to an ontology in order to support the automation of service related task.

While DAML-S defines service related concepts in relation to each other, it lacks the formal semantics to relate these concepts to the basic categories of philosophy, linguistics or human cognition. Typically for a domain ontology, there is no firm class or property hierarchy (most classes and properties are direct subclasses of the top level concept) and several relations take *Thing* as their domain or range. Part of the missing semantics is in the text of the document, while some are left to the reader or future work to decide.

We believe that this situation is not satisfactory: the level of commitment in DAML-S will need to be raised if it is to support the complex tasks put forward by the coalition (for a description of these tasks, see [Coa03, BCF+03]). Further axiomatization through alignment to a foundational ontology will help to exclude terminological and conceptual ambiguities due to unintended interpretations. This capacity will be critical if DAML-S is to be employed on a global scale, where the meaning of descriptions will need to be constantly negotiated.

Axiomatization is not without dangers of its own: it may lead to the creation of an overly restrictive, rigid ontology which would require a commitment that is difficult to achieve on a global scale (see [vEA02] for an analysis of the contradiction between the formality, sharing scope and stability of knowledge). However, we be-

lieve that this danger is mitigated by the design of DOLCE. While extensively researched and formalized, DOLCE is created with minimality in mind and includes only the most reusable and widely applicable upper-level categories [MBG+02]. DOLCE also calls for a careful isolation of the fundamental ontological options and their formal relationships and is built with modularization in mind. This means that DOLCE can avoid to become a single, monolithic upper-ontology that would be rejected by its users.

Note that DOLCE also allows us to observe minimality. In fact, our ontology is chiefly a combination of basic DOLCE and two extensions (an Ontology of Descriptions and an Ontology of Planning). To these existing ontologies less than 10 new concepts and 5 new properties were needed to be added to get to the core Ontology of Services.

3 Methodology

For the engineering of the Ontology of Services, we have chosen to follow a variation of ONIONS, the Ontologic Integration of Naive Sources methodology [GPS99]. ONIONS has been successfully applied in the past for various developments (e.g. an ontology of fishery services for the FAO of the UN). The methodology consists of the five steps shown below, which result in a new module (domain-specific extension) to a given foundational ontology (FO). Foundational ontologies such as DOLCE are explicitly designed as upper-level frameworks for analyzing, harmonizing and integrating existing ontologies and metadata standards [MBG+02].

1. **Re-engineering.** In the re-engineering phase, the sources are acquired and transformed in a uniform representation (data format).
2. **Integration.** In this step the sources are integrated in a logical sense. For example, distinctions between classes and instances are made, data types are harmonized etc.
3. **Alignment.** During alignment, concepts and relationships of the sources are characterized in terms of the concepts and relationships of a Foundational Ontology (FO). For example, at this stage classes described in the source ontologies are defined as subclasses of the most specific superclass available in the FO.
4. **Merging.** In the last step, concepts described in various sources are merged when they carry the same meaning with respect to the application scenario.

The sources in our case were the WSA document, DAML-S, parts of the Common Information Model (CIM) and the Ontology of Software Modules used within the ASSW.

Instead of direct alignment to the DOLCE foundational ontology, we decided to develop a Core Ontology of Services based on D&S (which is an extension to DOLCE) and aligned the sources to this ontology. This two-stage alignment is a common technique when the conceptual gap between the source ontologies and the foundational ontology is large. Also, formulated at a more generic level, one may expect the core ontology to be reusable later in other scenarios (e.g. our Ontology of Services may be reused for descriptions of purely commercial services. However, our sources are geared specifically towards information services, which means that the resulting ontology may lack some of the notions necessary for the matching and retrieval of commercial service offerings).

The remaining sections of this technical report is organized as follows. The Ontology of Descriptions (D&S) is introduced in Section 4. The Core Ontology of Services is presented in Section 5. Experiences with the alignment of the WSA document, DAML-S, and the Application Server's ontology are discussed in Sections 7 to 9, respectively.

4 Ontology of Description

The following subsections have been lifted from [GM] as background information on the ontology of descriptions.

4.1 Motivation

Foundational ontologies in WonderWeb are ontologies that contain a specification of domain-independent concepts and relations based on formal principles derived from linguistics, philosophy, and mathematics. Formal principles are needed to allow an explicit comparison between alternative ontologies. Examples of formal principles are spatio-temporal localization, topological closure, heterogeneity of parts, dependency on the intention of agents, etc. We refer to [MBG+02] for a detailed explanation.

While formalizing the principles governing physical objects or events is (quite) straightforward, intuition comes to odds when an ontology needs to be extended with *non-physical objects*, such as social institutions, organizations, plans, regulations, narratives, mental contents, schedules, parameters, diagnoses, etc. In fact, important fields of investigation have negated an ontological primitiveness to non-physical objects [Moo02], because they are taken to have meaning only in combination with some other entity, i.e. their intended meaning results from a statement. For example, a norm, a plan, or a social role are to be represented as a (set of) statement(s), not as concepts. This position is documented by the almost exclusive attention dedicated by many important theoretical frameworks (BDI agent model, theory of trust, situation calculus, formal context analysis), to states of affairs,

facts, beliefs, viewpoints, contexts, whose logical representation is set at the level of theories or models, not at the level of concepts or relations.

On the other hand, recent work (e.g. [Moo02]) addresses non-physical objects as first-order entities that can change, or that can be manipulated similarly to physical entities. This means that many relations and axioms that are valid for physical entities can be used for non-physical ones as well.

Here we support the position by which non-physical entities can be represented both as theories/models and as concepts with explicit reification rules, and we share the following motivations:

1. Technology and society are full of reifications, for example when we divide human experience into social, cultural, educational, political, religious, legal, economic, industrial, scientific or technological experiences

2. In realistic domains, specially in socially-intensive applications (e.g. law, finance, business, politics), a significant amount of terms convey concepts related to non-physical entities, and such concepts seem to be tightly interrelated

3. Interrelations between theories are notoriously difficult to be manipulated, then it would be an advantage to represent non-physical objects as instances of concepts instead of models satisfying some theory

4. For many domains of application, we are faced with partial theories and partial models that are explicated and/or used at various detail levels. Partiality and granularity are two more reasons to have some theories and models manipulated as first-order entities

5. Natural languages are able to reify whatever fragment of (usually informal) theories and models by simply creating or reusing a noun. Once linguistically reified, a theory or a model (either formal or informal) enters a life-cycle that allows agents to communicate even in presence of partial (or even no) information about the reified theory or model. The Web contains plenty of examples of such creatures: catalog subjects or topics, references to distributed resources, unstructured or semi-structured (but explicitly referenced) contents, such as plans, methods, regulations, formats, profiles, etc., and even linguistic elements and texts (taken independently from a particular physical encoding) can be considered a further example

6. Recent (still) unpublished work by one of the authors reports that more 25% of WordNet (v1.6) noun synsets [Fel98] can be formalised as non-physical object classes

In general, we feel entitled to say that representing ontological (reified) contexts is a difficult alternative to avoid, when so much domain-oriented and linguistic

categorisations involve reification. However, we also want to provide an explicit account of the contextual nature of non-physical entities and thus aim for a reification that accounts to some extent for the partial and hybrid structure of such entities.

From the logical viewpoint, any reification of theories and models provides a first order representation. From the ontological engineering viewpoint, a straightforward reification is not enough, since the elements resulting from reification must be framed within an ontology, possibly built according to a foundational ontology.

We also need specific reification rules for at least some distinct elements of a theory or a model. Moreover, from a practical viewpoint, the actual import of theories and models (when they are used as concepts) into an ontology requires not only reification rules, but also mapping and inheritance rules. This partial and hybrid transformation allows an easy grasp and manipulation of reified theories and models.

4.2 An Ontology of Descriptions and Situations

The Descriptions and Situations ontology (D&S) [GM03] is an attempt to define a theory that supports a first-order manipulation of theories and models, independently from the particular foundational ontology it is plugged in.

In general, D&S commits only to a widespread and very ancient ontological distinction between *flux*, or an unstructured world or context, and *logos*, or an intentionality. D&S is neutral with respect to realism issues, such as whether we conceive a structure because it is in the flux, or because it is in our intentionality [Mil02]. D&S as a representation mechanism makes no pretense in either direction. Hence, a flux can have as many inherent structure (parts, boundaries, qualities, etc.) as one might want to believe in or might claim to have discovered, but without a logos, a flux would have no description of that structure.

When logos is applied to the description of the flux, some *structure* emerges (this reflects the cognitive structuring cognitive process [Köh29]). The emerging structure is not necessarily equivalent to the actual structure.

Due to its neutrality with respect to realism, D&S can generalize the flux/logos distinction, in order to obtain an epistemological layering. Epistemological layering consists of assuming that any logical structure L_i (either formal or capable of being at least partly formalised) is built upon a flux-like structure that it describes according to a more abstract, logos-like theory T_i (either formal or capable of being at least partly formalised).

In other words, T_i describes what kind of ontological commitment L_i is supposed to have within the epistemological layer that is shared by the encoder of an ontology.

Epistemological layering reflects the so-called *figure-ground* shifting cognitive

process [Köh29]. Moreover, most assumption-makings in any domain of interest apply epistemological layering (several names have been used to refer to flux-like structures: tacit knowledge, context, substrate, etc.).

D&S implements reification rules for any T_i, called a *description*, and a basic framework for any L_i, called a *situation*[1], and for their elements.

Flux-like structures are not reified in D&S, but they result to be the structures that include all the (ground) logical dependencies of the components of a situation S classified within an ontology O, plus any additional elements that could be part of the ground context of S according to some encoder of O, but that are not inside O. A flux-like structure is called a *state of affairs* (SOA) in D&S.

4.3 Implementing the Ontology of Descriptions in DOLCE

DOLCE [MBG$^+$02] has four top categories: endurant (including object- and substance-like entities), perdurant (event- and state-like entities), quality (individual attributes), and abstracts (mainly conceptual "regions" for structuring attributes).

Within DOLCE, D&S is plugged in as follows. A situation is a (new) top category, a description is a non-physical endurant. Description is disjoint from situation. A description may be satisfied by a SOA. The satisfaction relation is reified in D&S as a first-order *referenced-by* relation. A description satisfied by a SOA is an *s-description*. A SOA satisfying a description is a situation.[2]

Concerning the reification of the elements of a theory, the descriptions that reify a selection rule on DOLCE regions (e.g. speed limit or visibility) are called *parameters*, the descriptions that reify a functional property of DOLCE endurants (e.g. citizen or judge) are called *functional roles*, and the descriptions that reify sequences of DOLCE perdurants (e.g. schedule or pathway) are called *courses*.

In D&S for DOLCE, descriptions have only other descriptions as parts. S-descriptions have courses, functional roles, and parameters as components. (See Fig. 2.) Between such components some relations hold: *modality-target* holding between functional roles and courses, and *requisite-for* holding between parameters and either functional roles or courses. Modality-target reifies the modal dependence between a functional property, and a sequence, while requisite-for reifies the logical dependence between a selection rule and either functional properties or sequences.

Situations and s-descriptions are systematically related. The basic relation is

[1] We are keeping these names for the historical reasons. Other intuitive names have been proposed so far, e.g. representation, conceptualisation, or schema for description, and setting, Gestalt, or configuration for situation.

[2] A situation can satisfy a (s-)description in many ways, so that we can build a taxonomy of satisfaction (referenced by) relations. This work, however, is outside the scope of this paper.

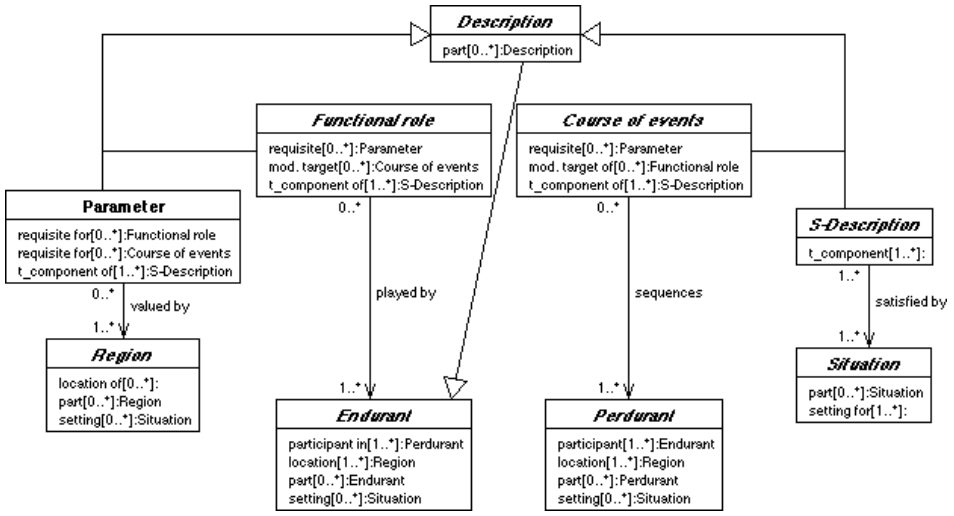

Figure 2: UML overview of the D&S ontology of descriptions

selects, and it reifies the instantiation relation between an individual in a model and a concept in a theory. Within DOLCE, *selects* relates components of an (s-) description to instances of DOLCE categories. Intuitively, selects(x,y) binds an individual y classified in a DOLCE category to a situation s that is referenced (satisfies) the s-description d that has x as a component. In particular: parameters are *valued-by* regions, f-roles *play* endurants, and courses *sequence* perdurants.

Examples of descriptions and situations include:

- A clinical condition (situation) has an associated diagnosis (s-description) made by some agent.
- A case in point (situation) is constrained by a certain norm (s-description)
- A murder (situation) has been reported by a witness (functional role) in a testimony (s-description)
- Information science as a topic (s-description) references the manipulation of data structures (situation), both as a pure or applied science (parent s-descriptions)
- A person (endurant) plays the role of judge (functional role) in the context of a constitutive Law (s-description)
- 40kmph (region) is the value for a speed limit (parameter) in the context of an accident (state of affairs) described as a speed excess case (situation) in an area covered by traffic Law (s-description)

D&S results to be a theory of ontological contexts because it is capable to describe various notions of context (physical and non-physical situations, topics, provisions, plans, assessments, beliefs, etc.) as first-order entities.

5 The Core Ontology of Services

The core ontology of services consists of a repeated application of the Ontology of Descriptions (D&S).

D&S provides reification rules for the three basic categories of DOLCE (region, endurants and perdurants), which are called parameters, roles and courses. D&S also defines a template, called S-Description (Situation Descriptions) for modelling non-physical contexts such as views, theories, beliefs, norms etc. An important distinction is made in D&S between (the elements of) descriptions and (elements of) a particular model, also called state-of-affairs (SOA): elements of a SOA (regions, endurants and perdurants) may play the parameters, roles and courses of a description, in which case the SOA is understood as a situation (case) for a particular description. However, the same SOA may be interpreted according to other, alternative descriptions. This captures an important feature of contexts, namely that multiple overlapping (or alternative) contexts may match the same world or model. For more information on D&S, we refer the reader to Section 4.

Service descriptions as non-physical contexts are ideally suited as applications of D&S. Descriptions of services can be considered as views from various perspectives on a series of activities that constitute the service for the various parties involved. In other words, service descriptions exhibit the same distinction between what is offered, expected or planned (descriptions, theories) and the elements that consist a particular model of the world.

Currently, we have considered five frequently occurring contexts regarding services, where each is a separate description of the same service in the D&S sense. More views may be added in the future when needs arise. Figure 3 shows their interrelationships.

1. **Service Offering (Description).** The Service Offering is the viewpoint of the legal entity providing the service. Much like commercial advertisements, the service offering may not describe entirely how the service will be carried out. This can also be considered as a proposal for a contract (agreement) for a service.

2. **Service Requirements (Description).** This is the counterpart of the offering in that it comprises the expectations of the requestor of the service. Requirements are often flexible, concerning only a subset of the tasks, roles and parameters of service activities (but might also contain others).

3. **Service Agreement (Description).** Once an agreement is reached between the provider and the requestor of the service, their joint understanding regarding the service may be described in a Service Agreement. Agreement means an understanding of the service as providing some value to the requestor, which may or may not be the same as the originally offered functionality of the service (in an extreme case, even doing nothing can be a service: consider the NOP command of machine language.) [3]

4. **Service Assessment (Description).** Typically, when an agreement is reached measures are taken to monitor, assess and control the execution of the service provided. Assessment concerns matching the service activities against the agreement.[4] Service assessment may be executed by a third party and may also involve aspects not even mentioned in the above three descriptions, e.g. the cleanliness of a hotel room may be checked by looking for dust on the TV sets. In the web services area, assessment is of particular concern to those interested in the management of web services.[5]

5. **Service Activities Description.** This is a description of the social conventions regarding the execution of a service, whether a written code of practice (ISO) or unwritten norm. This view is the basis for legal action once a service deviates from the norms in ways not foreseen in the agreement.

5.1 The Service Offering Description

In the following, we detail the structure of a Service Offering Description (see Figure 4). All other views are similar in nature.

The *Service Offering Description* is an S-Description, more precisely a Promise which has at least a single Service Task as temporary component.[6] A Task in DOLCE+ is a Course, which has only other tasks as temporary components and sequences at least one activity. A Task can also have a Situation as its precondition

[3] Independently from the fact that it may described, similarly to WSA we believe that in general an agreement (written or unwritten) between provider and requestor is necessary to talk of a service. Spam, or a dolphin saving someone in the middle of the ocean is not considered a service, no matter how useful it proves afterwards.

[4] In an ideal world such a function would be meaningless. In reality, contracts are incomplete, since it is difficult to imagine all possible outcomes flowing from the agreement. Also, violations and the resulting penalties are often accepted rather than adhering to the contract (a kind of control strategy).

[5] The WSA document, for example, stresses the manageability of web services as this is a key feature to companies interested in providing management platforms for web services. The CIM standard was also developed for creating a common format for exchanging information between management systems (Software designed to manage the IT assets of companies, including both their software and hardware environment).

[6] In the following, all categories and relations not printed in Italics are defined in DOLCE+, see Section 4.

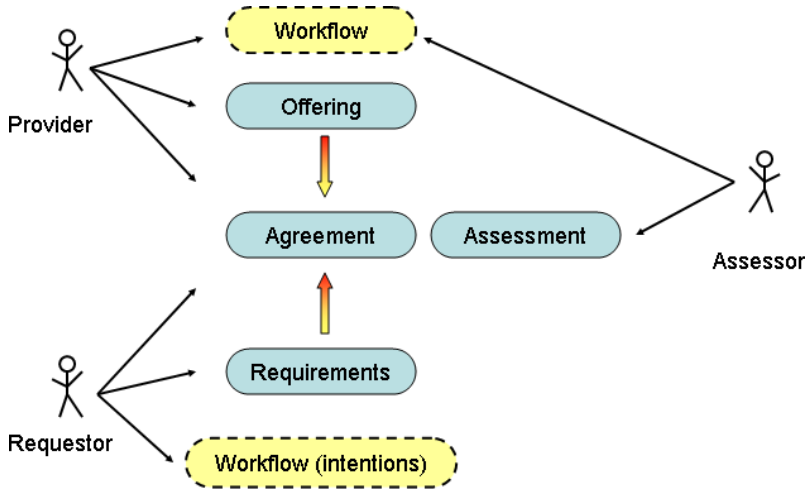

Figure 3: Relationships between the various views on a service.

or postcondition, which may or may not relate to (elements of) a situation for the description in which the Task is defined.[7]

We further define two disjoint subclasses of Task, *Service Task* and *Computational Task*. Service Tasks sequence only Service Activities and have only Service Tasks as temporary components. Similar statements hold for Computational Tasks. As we will see, the emergent distinction is that between tasks which require computational execution and work with information objects and tasks which involve physical objects.

A number of concepts from the Ontology of Planning are likely to be useful conjunction with the Core Ontology of Services. These include the division of tasks into elementary and complex tasks, and the construction of complex tasks from elementary ones. This part of the ontology is not detailed here, but can be consulted at `http://www.isib.cnr.it/infor/ontology/DOLCE.html`.

The chief difference between tasks and activities is that of between a plan and a particular execution of the plan: a plan represents possible sequences of execution. Examples of Computational Task are the reservation of a flight and the collection of payment, both in the sense of a transaction in an information system, even if it may be implemented in a number of ways. A Service Task can be flying the passenger (some passenger, not a particular one) to some destination. Again, this may be carried out in several ways.

In our ontology we also define a number of roles that are most commonly found in service descriptions. Two common agentive roles are introduced, namely *Requestor* and *Provider*. These are described as subclasses of the legally-constructed-

[7] We decided not to give different names to elements of the offering such as Service Offering Task. Unity criteria is given by the structure, i.e. the entire description.

person notion imported from a legal extension of DOLCE (Legally constructed persons are agentive functional roles played by socially constructed persons). In agreement with WSA, we conceive them as legal entities so that they can enter into agreement regarding a service. Examples are a passenger role (requestor of the booking service) and the role of the travel agency (provider of the service). We also conceive a third kind of agent role, namely that of the *Executor*. This can be used for modelling delegation.

Roles that are played by instruments of activities are called Instrumentality Roles in DOLCE. *Input* and *Output* are examples of such roles. *Computational Input* and *Computational Output* are kinds of input and output that are played only by information objects and only have exploitation within Computational Tasks.

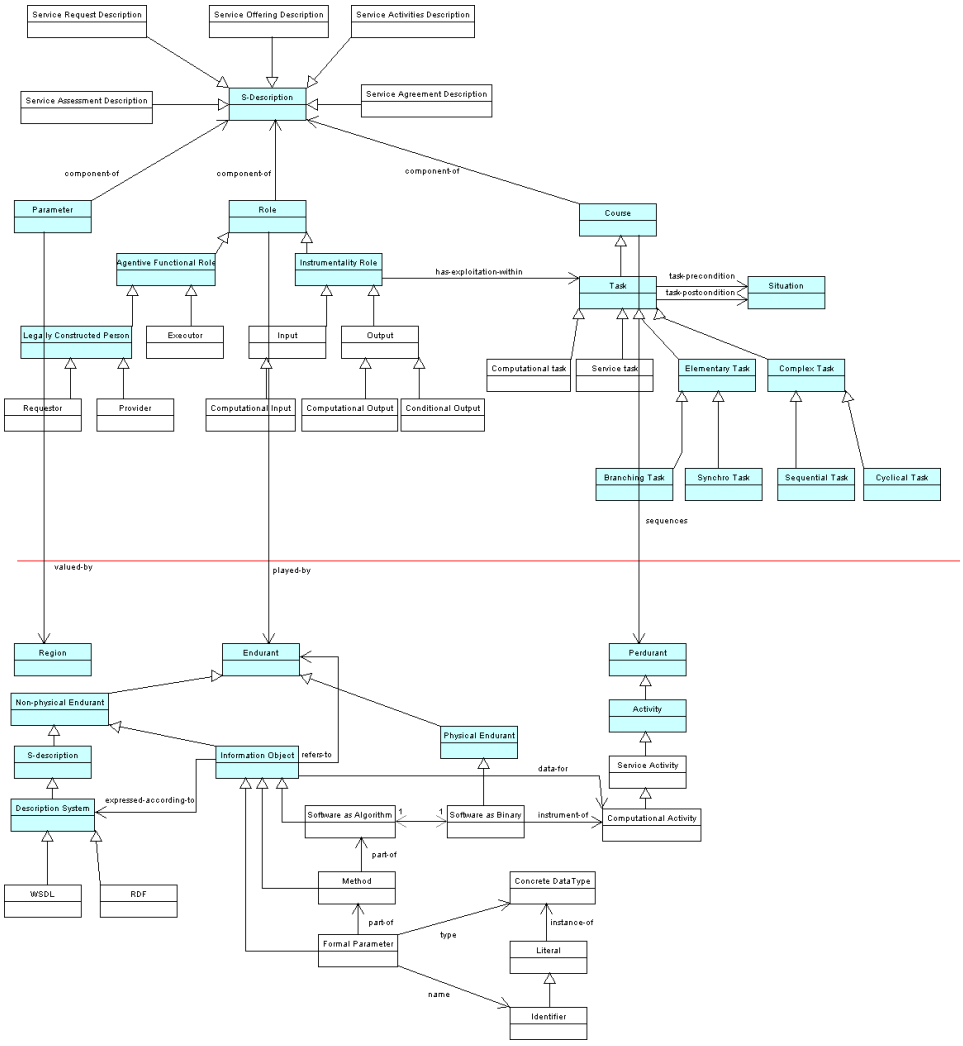

Figure 4: UML diagram of the Service Offering Description.

5.2 Service Situations

Our Service Offering Description introduced above stipulates the existence of a number entities in situations that satisfy the description. We also add some elements which may be useful in describing the settings of service executions.

A *Service Activity* is kind of Activity (a perdurant in DOLCE). A *Computational Activity* is a special kind of Service Activity which has only information objects or binary software as participants (Computational activity is another name for software as a perdurant). An example of a Service Activity would be flying Joe, a particular passenger, to his destination. An example of a Computational Activity would be the execution of the procedure that reserves a particular seat for a particular passenger.

Information Object is a non-physical endurant in DOLCE, which may be expressed according to a Description System. Examples of Description Systems are RDF or WSDL. As described in 6, *Software as Algorithm* is an information object, while *Software as Binary* represents its physical counterpart (more specifically, Software as Binary is said to be the instrument of a Computational Activity, while information objects are *data-for* the Computational Activity).

Assuming a procedural programming paradigm as common in the web services literature, Software as Algorithm is modelled as set of *Methods*. Methods in turn may have a number of *Parameters* as parts. Methods and Parameters are necessarily identified by names. Parameters must also have exactly one type.

We further introduce the minimal notions necessary for modelling information representation, partly based on earlier work on an ontology of communication and interpretation [GM03]. See Fig. 5 for an illustration.

In this example, Joe is a physical agent, but has a representation counterpart, namely the information object that is used to reference (identify) Joe in the software. The information object represents a meaning, an S-Description which may involve the entity in question. A Literal may extrinsically represent that information object, in which case the literal is said to be the name of the entity.

5.3 Axiomatization

$Service\_Offering\_Description(x) \rightarrow promise(x)$
$\forall x.Service\_Offering\_Description(x) \rightarrow$
$\exists y.temporary\_component(x, y) \wedge Service\_Task(y)$
$Service\_Requestor(x) \rightarrow Legally\_Constructed\_Person(x)$
$Service\_Provider(x) \rightarrow Legally\_Constructed\_Person(x)$
$Service\_Executor(x) \rightarrow agent\_role(x)$

$Service\_Input(x) \rightarrow non\_agentive\_functional\_role(x)$

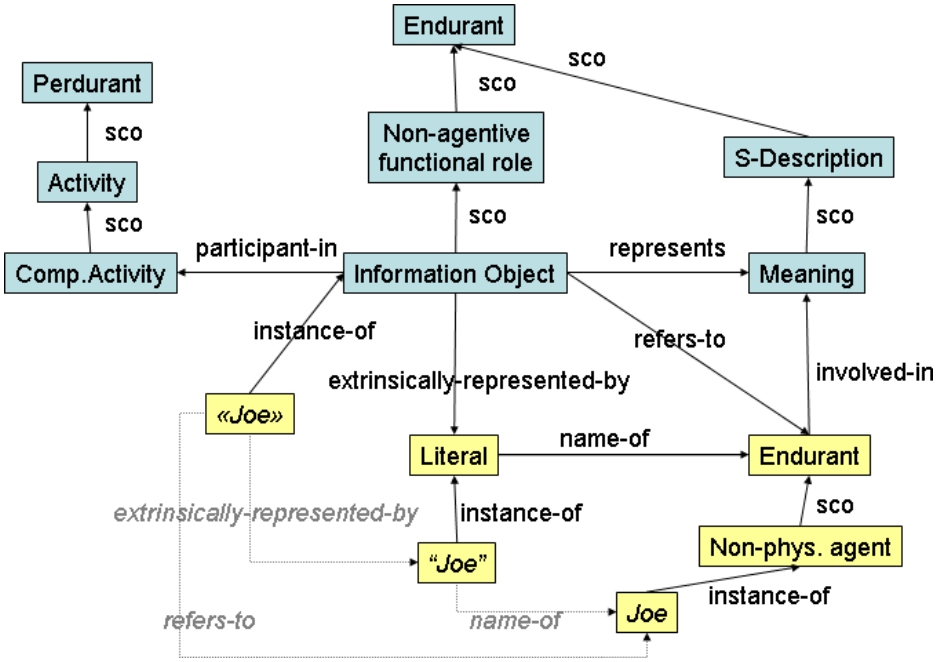

Figure 5: Modelling information representation.

$Computational\_Input(x) \rightarrow Service\_Input(x)$
$\forall x, y. Computational\_Input(x) \wedge played\_by(x, y) \rightarrow Information\_Object(y)$
$\forall x, y. Computational\_Input(x) \wedge has\_exploitation\_within(x, y) \rightarrow$
$Computational\_Task(y)$

$Service\_Output(x) \rightarrow non\_agentive\_functional\_role(x)$
$Computational\_Output(x) \rightarrow Service\_Output(x)$
$\forall x, y. Computational\_Output(x) \wedge played\_by(x, y) \rightarrow$
$Information\_Object(y)$
$\forall x, y. Computational\_Output(x) \wedge has\_exploitation\_within(x, y) \rightarrow$
$Computational\_Task(y)$

$Conditional\_Output(x) \rightarrow Service\_Output(x)$

$Computational\_Task(x) \rightarrow Task(x)$

$\forall x, y. Computational\_Task(x) \wedge sequences(x, y) \rightarrow$
$Computational\_Activity(y)$
$\forall x, y. Computational\_Task(x) \wedge temporary\_component(x, y) \rightarrow$
$Computational\_Task(y)$

$Service\_Task(x) \rightarrow Task(x)$
$\forall x, y.Service\_Task(x) \wedge sequences(x, y) \rightarrow Service\_Activity(y)$
$\forall x, y.Service\_Task(x) \wedge temporary\_component(x, y) \rightarrow Service\_Task(y)$

$Service\_Activity(x) \rightarrow Activity(x)$
$Computational\_Activity(x) \rightarrow Activity(x)$
$\forall x, y.Computational\_Activity(x) \wedge participant(x, y) \rightarrow$
$Information\_Object(y) \vee Software\_As\_Binary(y)$

$\neg(Computational\_Activity(x) \wedge Service\_Activity(x))$
$\neg(Computational\_Task(x) \wedge Service\_Task(x))$

$Software\_as\_Algorithm(x) \rightarrow Information\_Object(x)$
$Software\_as\_Binary(x) \rightarrow Physical\_Endurant(x)$

$Literal(x) \rightarrow Concrete\_Datatype(x)$
$Identifier(x) \rightarrow Literal(x)$

$Method(x) \rightarrow Information\_Object(x)$
$\forall x, y.Method(x) \wedge name(x, y) \rightarrow Identifier(y) \wedge \nexists zy \neq$
$z \wedge name(x, z) \wedge Identifier(z)$

$Formal\_Parameter(x) \rightarrow Information\_Object(x)$
$\forall x, y.Formal\_Parameter(x) \wedge name(x, y) \rightarrow Identifier(y) \wedge \neg\exists zy \neq$
$z \wedge name(x, z) \wedge Identifier(z)$
$\forall x, y.Formal\_Parameter(x) \wedge name(x, y) \rightarrow$
$Concrete\_Datatype(y) \wedge \neg\exists zy \neq z \wedge name(x, z) \wedge Concrete\_Datatype(z)$

$type(x, y) \rightarrow Property(x, y)$
$type(x, y) \rightarrow Formal\_Parameter(x)$
$type(x, y) \rightarrow Concrete\_Datatype(y)$
$type\_of(x, y) \rightarrow Property(x, y)$
$type\_of(x, y) \rightarrow Concrete\_Datatype(x)$
$type\_of(x, y) \rightarrow Formal\_Parameter(y)$
$type(x, y) \leftrightarrow type\_of(y, x)$

$extrinsically\_represented\_by(x, y) \rightarrow extrinsic\_relation(x, y)$
$extrinsically\_represented\_by(x, y) \rightarrow Information\_Object(x)$
$extrinsically\_represented\_by(x, y) \rightarrow Literal(y)$
$extrinsically\_represents(x, y) \rightarrow extrinsic\_relation(x, y)$
$extrinsically\_represents(x, y) \rightarrow Literal(x)$

$extrinsically\_represents(x, y) \rightarrow Information\_Object(y)$
$extrinsically\_represents(x, y) \leftrightarrow extrinsically\_represented\_by(y, x)$

$name\_of(x, y) \rightarrow extrinsic\_relation(x, y)$
$name\_of(x, y) \rightarrow Literal(x)$
$name\_of(x, y) \rightarrow Endurant(y)$
$name(x, y) \rightarrow extrinsic\_relation(x, y)$
$name(x, y) \rightarrow Endurant(x)$
$name(x, y) \rightarrow Literal(y)$
$name(x, y) \leftrightarrow name\_of(y, x)$

$data\_for(x, y) \rightarrow used\_in(x, y)$
$data\_for(x, y) \rightarrow Information\_Object(x)$
$data\_for(x, y) \rightarrow Computational\_Activity(y)$
$data(x, y) \rightarrow situation\_of\_use\_of(x, y)$
$data(x, y) \rightarrow Computational\_Activity(x)$
$data(x, y) \rightarrow Information\_Object(y)$
$data(x, y) \leftrightarrow data\_for(y, x)$

$task\_input(i, t) \leftrightarrow Task(t) \wedge Input(i) \wedge modality\_target(i, t)$
$task\_output(o, t) \leftrightarrow Task(t) \wedge Output(i) \wedge modality\_target(o, t)$

$NameOf(x, y) \leftrightarrow Literal(x) \wedge Entity(y) \wedge \exists z, w.Information\_Object(z) \wedge$
$Meaning(w) \wedge extrinsically\_represents(x, z) \wedge represents(z, w) \wedge$
$involves(w, y) \wedge refers\_to(z, y)$

$input\_for(io, a) \leftrightarrow$
$Information\_Object(io) \wedge Activity(a) \wedge \exists d, t, r.Serive\_Offering-$
$Description(d) \wedge Agentive\_Functional\_Role(r) \wedge Task(t) \wedge Input(r) \wedge$
$task\_input(r, t) \wedge sequences(t, a)$

$requestor\_in(e, a) \leftrightarrow$
$Endurant(e) \wedge Service\_Requestor(a) \wedge plays(e, a) \wedge participant\_in(e, a)$

$provider\_in(e, a) \leftrightarrow$
$Endurant(e) \wedge Service\_Provider(a) \wedge plays(e, a) \wedge participant\_in(e, a)$

$sequences(t, a) \wedge part(a, b) \rightarrow sequences(t, b)$

$participant - in(e, p) \wedge setting(p, s) \rightarrow setting(e, s)$

6 Defining web services: On the border of Infolandia

The greatest obstacle in conceptualizing web services seems to be the name itself, which is severely overloaded in meaning. Here are just some of the various definitions found in the literature:

1. A web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols [BCF+03].

2. A web service is viewed as an abstract notion that must be implemented by a concrete agent. The agent is the physical entity (a piece of software) that sends and receives messages, while the service is the abstract set of functionality that is provided. To illustrate this distinction, you might implement a particular web service using one agent one day (perhaps written in one programming language), and a different agent the next day (perhaps written in a different programming language). Although the agent may have changed, the web service remains the same (also from [BCF+03], although in clear contradiction to the previous def.)

3. A service is an active program or a software component in a given environment that provides and manages access to a resource that is essential for the function of other entities in the environment. A web service is a service that abides by a specific framework to offer its services. The framework provides the means to describe and discover the service, audit its service offering, and integrate the service with other services to offer higher-level services.[8]

4. Loosely speaking, a web service is a piece of functionality (an object, a component, an application, a database call) that can be invoked over a network using a predefined syntax.[9]

5. First of all, we start with an application that you want others to use. That is, you have a piece of software that initiates or accepts business transactions, provides or updates enterprise information, or perhaps manages the very systems and processes that make your business run. You may want to make this accessible to people in other parts of your organization, or a business partner, or a supplier, or a customer. We're really thinking here about software-to-software communication rather than the person-sitting-at-a-browser-talking-to-server-software situation, though it turns out that web services can be used there as well.[10]

[8] cf. `http://www.informit.com` under "Web Development", "Web services".

[9] cf. `http://www.informit.com`, Article "Web Services Part 3: What Are Web Services" by Alex Nghiem.

[10] cf. `http://searchwebservices.techtarget.com`, definition of web services

6. Among the most important Web resources are those that provide services. By "service" we mean Web sites that do not merely provide static information but allow one to effect some action or change in the world, such as the sale of a product or the control of a physical device. The Semantic Web should enable users to locate, select, employ, compose, and monitor Web-based services automatically... Any Web-accessible program/sensor/device that is declared as a service will be regarded as a service. DAML-S does not preclude declaring simple, static Web pages to be services. But our primary motivation in defining DAML-S has been to support more complex tasks like those described above. [Coa03]

These definitions call one of the following (or both, as in the case of WSA) a web service:

1. An information system, invokeable using particular technologies such as XML, i.e. accessible through the Web. This is often confused with the functionality attributed to the service, even though functionality of a tool is contingent on usefulness in a particular situation.[11]
2. Some functionality (service) provided and a task to be fulfilled. This task is external to the software, e.g. a business transaction.
3. An interface to a software or heterogeneous system, which makes it web accessible. Having a publicly available description of a service is often considered a requirement to call it a web service. As a consequence, this view often goes as far as equating the web service to (the description of) an interface.

We have to separate these concepts in order to modularize our descriptions of services. It seems that at the heart of the entanglement between software, functionality and interfaces lies a disregard to the fact that web services exist on the boundary of the world inside an information system (Infolandia) and the outside world:

The scope of "Web services" as that term is used by this working group is somewhat different. It encompasses not only the Web and REST Web services whose purpose is to create, retrieve, update, and delete information resources but extends the scope to consider services that perform an arbitrarily complex set of operations on resources that may not be "on the Web." Although the distinctions here are murky and controversial, a "web service" invocation may lead to services being performed by people, physical objects being moved around (e.g. books delivered). [BCF+03]

[11] Similar phenomena exist with real world objects: a hammer becomes a "tool" instead of an artifact when it is in the hands of someone who knows how to use it. Otherwise, it's an amount of matter.

Thus web services carry out computational activities to *support* a service. But can we call the software a service? We believe that is not the case: usefulness, which is an essential property of a service, arises from the entire process involving real world as well as computational activities. In the case of a flight booking service, the customer of the service values the fact that as a result of the service, he will be able to transport himself to one place or another. The fact that part of the execution involves an interaction between the travel agent and the customer through an information system (e.g. a WWW site) is a mere implementation aspect from the customer point of view. This is not to say that there cannot exist services which concern purely information objects, e.g. the transformation of some data from one from to another. Most services offered via the Web, however, will not be pure information services.

The curious positioning of web services holds a particular challenge for ontological modelling. Descriptions of web services are, in fact, descriptions of two parallel worlds. In Infolandia, the world consist of software manipulating (representations of) information objects. Activities are sequenced by computational processes. Meantime in the real world passengers and airplanes are flying to their destinations. The connection between these worlds is simply that some of the information objects in Infolandia are symbols of (or identifiers for) real world objects. Also, computational activities comprise part of the service execution in the real world. For example, a booking needs to be entered by the travel agent into an information system, so that the airline would know which passengers to allow on the plane.

Since software stands in between the information and the real world, it stretches the categories of foundational ontologies.[12] Upon close inspection, it seems that the term software is also heavily inflicted by polysemy and refers to at least four different concepts:

1. An algorithm. An algorithm is like a tune in music, distinct from its notations or executions. Algorithm is an endurant in DOLCE terms.

2. The encoding of an algorithm in some kind of representation, e.g. binary or Java code. Encoding can be either in mind, on paper or any other form. This is software as information object, which is also an endurant.

3. Static implementation of software, which is a file on someone's computer with the executable code. Different from the previous category in that it's a directly exploitable form. This kind of software is a perdurant or 4D object[13].

[12] The problem is similar to modelling communication, which occurs in three layers: 1) meaning 2) symbols, expressions 3) physical signals transmitted through a channel. The first two aspects are logical, while the last one is physical, yet part of the same process.

[13] Strictly speaking software is a 4D object: while someone can sit on a chair at a certain point in time, it is not possible to make sense of software at a given point in time. 4D objects are not yet covered by DOLCE.

4. The running system, which is the result of an execution. This is the form of software which manifests itself in the form electrical signals rising and dropping, the screen flickering and sound coming out the machine. This form of software is a physical perdurant or 4D object.

The first two items represent software as a product, while the latter two refer to the process nature of software.[14] The two seem just as inseparable as the wave and particle nature of light: without hardware in the physical world, no software would exist. In other words, perdurancy mutually depends on endurancy: for each state of a perdurant (software), there is a state of an endurant (hardware) reflecting that perdurant. Nevertheless, when we want to separate the two aspects of software in our descriptions, we will talk about Software-as-Perdurant and Software-as-Endurant.

7 Alignment of the Web Services Architecture

The Web Services Architecture (WSA) document is a work of the similarly named working group of the W3C, whose membership is almost exclusively comprised by industry representatives. The document is an effort by the W3C to create a conceptual framework of web services based, which matches the requirements collected in [ABFG02]. The document is also input to other web services related activities at the W3C, namely the XML Protocol Working Group (responsible, among others, for SOAP), the Web Services Description Working Group (working on WSDL) and the Web Services Choreography Working Group (working on service composition). The WSA is still a work in progress[15], which means that our comments may be outdated.

In general, the document shows a great deal of confusion over the definition of a web service (see also Section 6). The current defines the web service as a software system and requires that web services are identified by a URI and their public interfaces and bindings are defined and described using XML. However, the authors themselves express doubts whether it's truly required for a web service to have a public description. The notion of binding is left undefined. Mentioning XML as base technology is also somewhat awkward, considering that it only concerns representation (ASCII or Unicode is then also a requirement).[16]

Only one section later, in contradiction with the earlier definition, a web service is called an abstract notion that is implemented by an agent (a software). While it's not explained what this abstract notion is, the document notes that the purpose of

[14] Similar bipolar effect characterizes the difference between service and product in the commercial world. Products can be viewed as a service: if someone buys a house for lifetime rental, what he actually buys is the right to live there for the end of his life.

[15] W3C Working Draft of May 14, 2003

[16] The intention of the definition is to stress the interoperability requirements for web services . The document tries to be neutral with regard to more web-service-specific protocols.

a web service is to provide some functionality on behalf of its owner.[17] Further, in Section 1.6.2, the document returns to the original definition, when doubts are expressed in the comments whether the web service is the external code or an interface to some external code.

Besides notes on the architecture, the document also provides a collection of "Core Concepts and Relationships". Unfortunately, this is only available in text and pictures. (For that reason, we did not perform the actual physical alignment.)

Here we go through the major concepts, skipping features of the entire architecture, acts and concepts related to the management of web services.

Skipped: authentication, choreography description language, correlation, discovery, discovery service, feature, identifier, intermediary, life cycle, management capability, management configuration, management event, manager, manageable element, manageability interface, management metric, message exchange pattern, message header, message description language, message identifier, reliable messaging, representation, resource, SOAP, WSDL.

Agent A program, i.e. a software acting on behalf of a legal entity. A deployed element, i.e. physical.
sameAs SoftwareAsEndurant and it plays computational agent role

Choreography A choreography is a set of possible interactions between a set of services.
A choreography is thus another description, which operates on the union of the regions, endurants and perdurants referenced by the individual service descriptions. A choreography expresses only possible interactions, and therefore it is distinct from a composite service, i.e. a possible realization of interacting services.

Deployed element Deployed element is the collective name for physical objects. Agents, services and descriptions are mentioned as kinds of deployed elements. Deployed element is introduced also as a unit of manageability.

Legal entity Same as our definition.

Message A "unit of interaction between agents".
Message is a functional role in communication played exclusively by information objects. (Pigeons carrying letters seem to be excluded)

Message Sender, Message Receiver Conceived as kinds of agents.
We model sender and receiver as functional roles in communication.

[17] "The provider entity is the legal entity that provides an appropriate agent to implement a particular service." How does one determine whether an agent is appropriate before an agreement is reached over the service? General feeling is that the industry community thinks of a web service as an extra interface to an existing line-of-business system, i.e. functionality is engrained.

Service Again a new definition, emphasizing the process nature of a service and the agreement needed: "A service is a set of actions that form a coherent whole from the point of view of service providers and service requesters."
If we disregard the universal, objectivist view of a service, this seems to be close to the set of tasks performed by a service or the entire description.

Service Description A "set of documents" that describe the interface to and semantics of a service.

If set of documents is meant in a representation-independent way, its akin to an information object representing the service (offering) description.

Service Provider, Service Requester Conceived as kinds of agents.
We model providers and requesters as functional roles in some description of a service.

Service Semantics "The semantics of a service is the contract between the service provider and the service requester that expresses the effect of invoking the service."
Clearly, this is the Service Agreement Description.

Service Task "A service task is a unit of activity associated with a service. It is denoted by a pair: a goal and an action; the goal denotes the intended effect of the task and the action denotes the process by which the goal is achieved."
Matches the DOLCE notion of a task.

8 Alignment of DAML-S

DAML-S divides information about a web service into three kinds of descriptions: profiles, processes and groundings. The reason behind this separation are the different functions these descriptions are designed to support. Profiles are primarily intended for discovery and matching of service offerings and requests, therefore profiles contain metadata about the service (classification, ratings, source) as well as inputs, outputs, preconditions and effects of the entire service. Process descriptions, on the other hand, support the composition of web services by describing the IOPEs of individual atomic services that may be identified within the service and valid sequences of executions. Lastly, grounding concerns the information necessary to invoke a web service over the internet. (All three kinds of descriptions are meant for machine processing.)

The goal of all modularizations is a separation of concerns. Given some division of concerns, a modularization is optimal if it reduces the need for links between modules in order to attend to those concerns (overlapping or cross-cutting concerns are problematic as there is a need to duplicate information, see the difficulty of maintaining consistency between IOPEs in the process and the profile).

This suggests that related information, which is expected to be used in conjunction with the same concern, should be allocated to the same module. Without a history of usage of web services, it is not known at this point how the information available in web service descriptions would be used and therefore it is difficult to tell if the divisions in DAML-S are indeed the optimal ones.

Our ontology suggests one important dimension for modularization: the distinction between elements of the description (a plan) and a situation (its execution). However, we leave further modularization dependent on future use cases for our descriptions (on the technical side, we are also waiting for a more versatile modularization mechanism than namespaces).

Although the definition of a service is ambiguous even in the natural text description of DAML-S, for the sake of argument we considered an *daml-s:Service* as a Service Offering Description, which has the *ServiceProfile* and *ServiceModel* (also Service Offering Descriptions) as parts. Actors in the *ServiceProfile* are aligned as Agentive Functional Roles . The *ServiceModel* concept was aligned to our Service Task concept, while the individual control constructs were mapped to task components provided by the Ontology of Plans.

In the Core Ontology of Services, the notions of Inputs and Outputs were modelled as Non-Agentive Functional Roles and not as relations in DAML-S. Nevertheless, alignment was possible by means of a composed relationship. On the other hand, the notion of preconditions and effects are inherited from the Ontology of Plans (task-precondition and task-postcondition) where they are modelled as Situations.

As it was not related to the focus of work, we omitted the alignment of the particular grounding ontology for WSDL [CCMW03]. Nevertheless, the notion of *Software* is present in the Core Ontology of Services as *Information Object* that can be expressed according to any number description systems.[18] WSDL could be considered as such a description system and modelled to the extent required to express groundings.

To the observer, our ontology might seem to be more verbose than DAML-S. In fact, we decompose many of the relationships in DAML-S, such as the link between endurants and their representation in information systems. We also decompose the grounding relation of DAML-S between processes and software implementations. Our goal in these decompositions is to find semantically distinct building blocks of these relationships and thus reconstruct semantics. In effect, DAML-S relationships may be easily recomposed from these blocks. For example, we may introduce a composed relationship between information objects and tasks, which says that if an information object *plays* input and that input *has exploitation within* a given task, we might say that such an information object is *input-for* that

[18] An alternative, more refined representation we considered was to model *Software* as an S-Description, in the sense of an abstract algorithm.

task, mimicking the similar relationship in DAML-S.

8.1 Illustrated example

In this Section we show how the semantics of the Congo example of DAML-S could be represented by our Core Ontology of Services. For the purposes of this demonstration, we shortened the example to the part described in [MBD+03].

We begin with the Service Offering Description proposed by Congo Inc., called CongoBuyOffering. CongoBuyOffering has a number of functional roles and tasks as parts.

$CongoBuyOffering(x) \rightarrow Service\_Offering\_Description(x)$
$CongoCustomer(x) \rightarrow Service\_Requestor(x)$
$\forall x, y. CongoCustomer(x) \wedge temporary\_component\_of(x, y) \rightarrow$
$CongoBuyOffering(y)$
$CongoProvider(x) \rightarrow Service\_Provider(x)$
$\forall x, y. CongoProvider(x) \wedge temporary\_component\_of(x, y) \rightarrow$
$CongoBuyOffering(y)$

In all situations, CongoInc necessarily plays the role of the provider (a role restriction).

$agentive\_physical\_object(CongoInc)$
$\forall x, y. CongoProvider(x) \wedge played\_by(x, y) \rightarrow y = CongoInc$

LocateBook and BuyBook are elementary computational tasks.

$LocateBook(x) \rightarrow Computational\_Task(x)$
$LocateBook(x) \rightarrow elementary\_task(x)$
$BuyBook(x) \rightarrow Computational\_Task(x)$
$BuyBook(x) \rightarrow elementary\_task(x)$

ExpandedCongoBuy is a complex service task, which has LocateBook and BuyBook as parts and is itself a temporary component of the offering. It is inferred that LocateBook and BuyBook are also temporary components.

$ExpandedCongoBuy(x) \rightarrow Service\_Task(x)$
$ExpandedCongoBuy(x) \rightarrow complex\_Task(x)$
$\forall x, y. LocateBook(x) \wedge part\_of(x, y) \rightarrow ExpandedCongoBuy(y)$
$\forall x, y. BuyBook(x) \wedge part\_of(x, y) \rightarrow ExpandedCongoBuy(y)$
$\forall x, y. ExpandedCongoBuy(x) \wedge temporary\_component\_of(x, y) \rightarrow$
$CongoBuyOffering(y)$

BookToLocate is a computational input to LocateBook. DescriptionOutput and CatalogueBookOutput are conditional computational outputs of LocateBook.

$BookToLocate(x) \rightarrow Computational\_Input(x)$
$\forall x, y. BookToLocate(x) \wedge modality\_target(x, y) \rightarrow LocateBook(y)$
$DescriptionOutput(x) \rightarrow Conditional\_Output(x)$
$DescriptionOutput(x) \rightarrow Computational\_Output(x)$
$CatalogueBookOutput(x) \rightarrow Conditional\_Output(x)$
$CatalogueBookOutput(x) \rightarrow Computational\_Output(x)$
$\forall x, y. DescriptionOutput(x) \wedge modality\_target(x, y) \rightarrow LocateBook(y)$
$\forall x, y. CatalogueBookOutput(x) \wedge modality\_target(x, y) \rightarrow LocateBook(y)$

BookToLocate is played by information objects in RDF that reference a book (Role playing can be similarly restricted for the outputs of BookToLocate).

$BookDescription(x) \rightarrow Information\_Object(x)$
$language(RDF)$
$\forall x, y. BookDescription(x) \wedge expressed\_according\_to(x, y) \rightarrow y = RDF$
$Book(x) \rightarrow Physical\_Endurant(x)$
$\forall x, y. BookDescription(x) \wedge refers\_to(x, y) \rightarrow Book(y)$
$\forall x, y. BookToLocate(x) \wedge played\_by(x, y) \rightarrow BookDescription(y)$

Next, we model an actual sale of a book. We show that this can be understood as a situation for the above description by mapping between elements of the setting and the service offering description. Note that this implies, for example, that CongoInc is necessarily participating in this sale as the provider.

$Situation(CongoSale)$
$CongoBuyOffering(cbo)$
$satisfies(CongoSale, cbo)$

Joe is a CongoInc customer, who participates in the activity.

$natural\_person(Joe)$
$CongoCustomer(cc)$
$plays(Joe, cc)$
$participant\_in(Joe, BuyingWinnieThePooh)$

BookObject is an information object (document), which refers to WinnieThePooh, a book that the customer would like to find.

$Book(WinnieThePooh)$

$Literal(\text{"}WinnieThePooh\text{"})$
$name\_of(WinnieThePooh, \text{"}WinnieThePooh\text{"})$
$part\_of(WinnieThePooh, CongoSale)$
$BookDescription(BookObject)$
$refers\_to(BookObject, WinnieThePooh)$
$BookToLocate(WinnieThePooh)$
$plays(BookObject, WinnieThePooh)$
$part(BookObject, CongoSale)$

BuyingWinnieThePooh is the actual activity that is performed in this sale according to the task description. LocatingWinnieThePooh is a computational part of the activity that is carried out to locate the book. The BookObject is data for this activity.

$Service\_Activity(BuyingWinnieThePooh)$
$Computational\_Activity(LocatingWinnieThePooh)$
$part\_of(LocatingWinnieThePooh, BuyingWinnieThePooh)$
$setting(BuyingWinnieThePooh, CongoSale)$
$ExpandedCongoBuy(ecb)$
$sequences(ecb, BuyingWinnieThePooh)$
$data\_for(BookObject, LocatingWinnieThePooh)$

We don't capture that Joe provides the information object, i.e. the book to locate. We do capture that the information object references a book, and we could capture as a precondition that Joe wants book. We could also describe the effect: Joe has a book.

9 Alignment of the Application Server's ontology

9.1 Original Ontology

The Application Server for the Semantic Web uses an ontology for software module and API discovery, manual classification of software modules and for implementation tasks [OVMS03]. During its design we tried to stay as close as possible to DAML-S (cf. Section 8) for it is an accepted standard that has been investigated for a long time and has a sound basis [OSRV03].

Although DAML-S serves as a good starting point for our ontology, the main difficulty was in the type of software entities to be described. While DAML-S describes web services, our goal is to describe software modules and their APIs. As a result some parts of DAML-S were not reusable. In the Appendix we present all the subontologies in DAML-S in comparison to ours before the alignment. What

we will achieve in the next subsection is the alignment from the generic level, represented by DOLCE, D&S and the Core Ontology of Services, to the intermediate and domain level.

The *Implementation* subontology is primarily used to facilitate component discovery for the client and of particular importance as it introduces several new concepts. Its terminology is shown below.

Software Module Speaking in terms of the object-oriented paradigm, a software module is an object revealing an Application Programming Interface (API). A software module fulfills complex computational tasks. Examples: ontology store, inference engine.

Component Software module that is deployed to the Application Server for the Semantic Web [19].

System Component Component providing functionality for the Application Server for the Semantic Web itself, e.g. the registry.

Functional Component Component that is of interest to the client and can be discovered. Ontology-related software modules become functional components by making them deployable, e.g. RDF stores.

External Module An external module cannot be deployed directly as it may be programmed in a different language, live on a different computing platform, etc. It equals a functional component from a client perspective. This is achieved by having a proxy component deployed that relays communication to the external module.

Proxy Component Special type of functional component that manages the communication to an external module. Examples are proxy components for inference engines, like FaCT.

Interceptor Software that monitors requests and modifies them. Examples: transaction or semantic interoperation interceptor.

Surrogate Software embedded in the client application. It offers the same API as a particular component and relays communication to it. Meant for ease of use in the ASSW scenario, similar to stubs in CORBA.

9.2 Aligning the taxonomy

In a first step, we strive to align the terminology in the subsection above. Figure 6 sketches an overview before we detail the concept's axioms in the following paragraphs.

[19] We use the word deployment as the process of registering, possibly initializing and starting a component to the Microkernel.

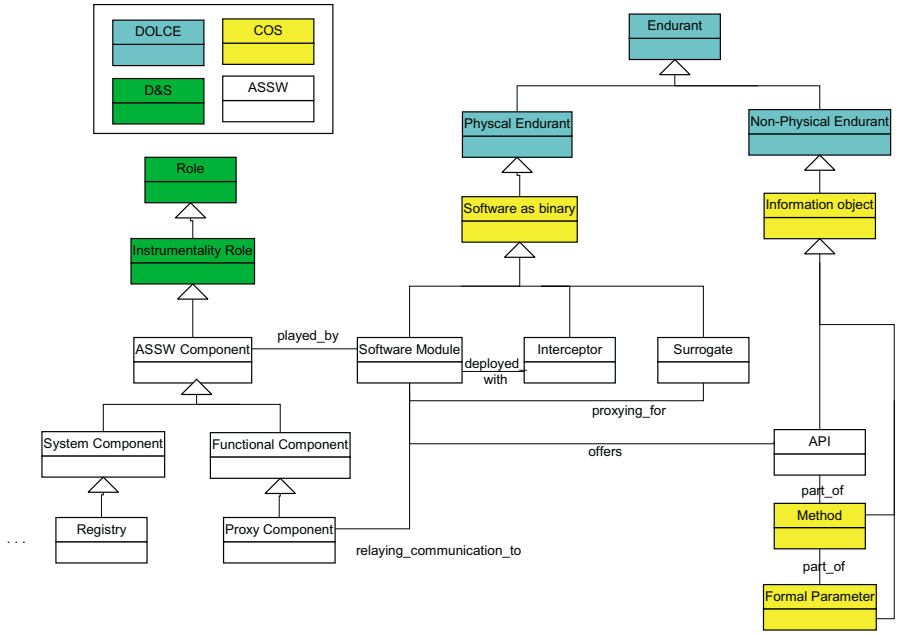

Figure 6: Alignment of the ASSW's concepts

Software Module, Interceptor and Surrogate become subconcepts of Software-as-binary. A Software module offers an API which in turn is subconcept of Information Object. An API consists of Methods and a Method may have Formal Parameters. Software Modules are deployed with an Interceptor and Surrogates proxy for Software Modules on the client side.

$$Software\_Module(x) \rightarrow Software\_as\_binary(x)$$
$$Interceptor(x) \rightarrow Software\_as\_binary(x)$$
$$Surrogate(x) \rightarrow Software\_as\_binary(x)$$
$$API(x) \rightarrow Information\_object(x)$$
$$offers(x, y) \rightarrow Software\_Module(x)$$
$$offers(x, y) \rightarrow API(y)$$
$$deployed\_with(x, y) \rightarrow Software\_Module(x)$$
$$deployed\_with(x, y) \rightarrow Interceptor(y)$$
$$proxying\_for(x, y) \rightarrow Surrogate(x)$$
$$proxying\_for(x, y) \rightarrow Software\_Module(y)$$

While the conceptualization above is quite generic, Software Modules can become Components in the Application Server for the Semantic Web setting (formalizing the specializations of Component is straightforward). This behavior shows a clear contextual nature and, thus, we model an ASSW Component as a role played by a Software Module. The most prominent example for that is an Ontology Store

Software Module which is a first order entity but can be both the registry (i.e. a System Component) and a Functional Component within the Application Server.

$ASSW\_Component(x) \rightarrow Instrumentality\_Role(x)$
$\forall x, y. ASSW\_Component(x) \wedge played\_by(x, y) \rightarrow Software\_Module(y)$
$Functional\_Component(x) \rightarrow ASSW\_Component(x)$
$Proxy\_Component(x) \rightarrow Functional\_Component(x)$
$System\_Component(x) \rightarrow ASSW\_Component(x)$
$Registry(x) \rightarrow System\_Component(x)$
...

Note that we do not list all specializations of System Component here (Registry, Association Management, Component Loader, Cascading Component, etc.). Note also, that there is no need to model External Modules. It is enough to formalize Proxy Component as a role that relays communication to a Software Module.

$relaying\_communication\_to(x, y) \rightarrow Proxy\_Component(x)$
$relaying\_communication\_to(x, y) \rightarrow Software\_Module(y)$

9.3 API Descriptions

After aligning the terminology we would like to capture the intuition that is common in both DAML-S and ASSW, namely that there are semantic descriptions of software (describing functionality or tasks) and syntactic descriptions of software (describing parts of software as an object). Hence we come up with a new kind of description in the D&S sense, called APIDescription (cf. Figure 7).

In fact, we formalize a whole hierarchy of APIDescriptions as domain knowledge. E.g., in the Semantic Web domain, StoreAPIDescription along subconcepts like RDFStoreAPIDescription or OntologyStoreAPIDescription. What is common to all APIDescriptions is that there has to be a role ASSW Component played by Software Module and the ASSW Component has exploitation within at least one Computational Task. The last relation is refined for specializations of APIDescriptions, e.g. in an RDFStoreAPIDescription the role of a Functional Component has exploitation within a StoreTriple Computational Task etc.

$StoreAPIDescription(x) \rightarrow APIDescription(x)$
$RDFStoreAPIDescription(x) \rightarrow StoreAPIDescription(x)$
$OntologyStoreAPIDescription(x) \rightarrow StoreAPIDescription(x)$
...
$\forall x. APIDescription(x) \rightarrow \exists y, z, t. component\_of(x, y) \wedge$
$ASSW\_Component(y) \wedge played\_by(y, z) \wedge SoftwareModule(z) \wedge$
$has\_exploitation\_within(y, t) \wedge computational\_task(t)$

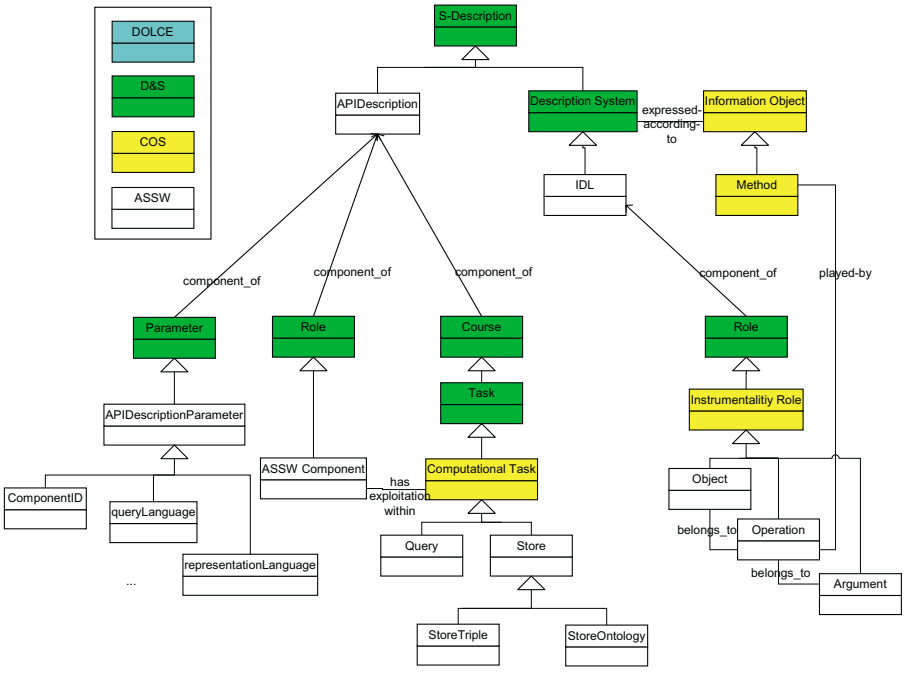

Figure 7: API Description

...
$\forall x.RDFStoreAPIDescription(x) \rightarrow$
$\exists y, z, t.component\_of(x, y) \wedge Functional\_Component(y) \wedge played\_by(y, z) \wedge$
$SoftwareModule(z) \wedge has\_exploitation\_within(y, t) \wedge StoreTriple(t)$
...

9.3.1 Roles

The new roles introduced in the subsection above are relevant for the API Description. So-called ASSW Components and specializations are played by Software Modules (cf. Figure 6). Every ASSW Component has exploitation within a Computational Task.

9.3.2 Courses

As depicted in Figure 7 we use Computational Task which is part of the Core Ontology of Services and subconcept of DOLCE's Course. We define new, domain dependent, specializations thereof. In the example, we come up with Semantic Web related Computational Task like StoreTriple or StoreOntology. They become components of the API Description and have exploitation within the ASSW Component role which are ultimately played by Software Modules.

$Store(x) \rightarrow computational\_task(x)$
$StoreTriple(x) \rightarrow Store(x)$
$StoreOntology(x) \rightarrow Store(x)$
...
$Query(x) \rightarrow computational\_task(x)$

The other way around, it is important to model which Method fulfills the Computational Tasks mentioned above. Therefore we have to define a new relation 'fulfills' between Information Object and Computational Task independent of the APIDescription.

$fulfills(x, y) \rightarrow Information\_Object(x)$
$fulfills(x, y) \rightarrow Computational\_Task(x)$

9.3.3 Parameters

When a Software Module is deployed to the Application Server for the Semantic Web, it automatically gains several attributes, most prominently a ComponentID. Such properties do not belong to the software module but show a clear context dependency. Hence, we model them as new parameters that are component of the APIDescription (cf. Figure 7).

$ComponentID(x) \rightarrow APIDescriptionParameter(x)$
$\forall x.ComponentID(x) \rightarrow \exists y.APIDescription(y) \wedge component\_of(y, x)$
...

In addition, specializations of the APIDescription may have several domain-dependent properties. E.g., an StoreAPIDescription may have a parameter representationLanguage or queryLanguage. [Per02] gives a nice overview of different Semantic Web software modules and their characteristika. Such relations have to axiomatized accordingly, e.g.

$queryLanguage(x) \rightarrow APIDescriptionParameter(x)$
$\forall x.queryLanguage(x) \rightarrow$
$\exists y.StoreAPIDescription(y) \wedge component\_of(y, x)$
...

Figure 7 sketches the newly introduced parameter called APIDescriptionParameter which can be component of APIDescriptions only. Note that an APIDescription is not expected to have a certain number of parameters as component. They are optional altogether.

9.4 IDL Descriptions

For the syntactic descriptions of software we come up with a new kind of description called IDLDescription. For this purpose we formalized the terminology of IDL (Interface Description Language [Gro02]), viz. Object, Operation, Argument etc., as instrumentality roles. The idea is that such roles are played by information objects, e.g. Object is played by Software Module and Operation is played by Method.

The general idea is already featured in the Core Ontology of Services where Description Systems are introduced as subconcept of D&S's description. Information Objects, which are non physical Endurants, are expressed according to such a Description System. Examples would be RDF or the aforementioned IDL.

$IDLDescription(x) \rightarrow APIDescription(x)$
$\forall x.IDLDescription(x) \rightarrow \exists y.component\_of(x,y) \wedge Object(y)$
$\forall x.IDLDescription(x) \rightarrow \exists y.component\_of(x,y) \wedge Operation(y)$
$\forall x.IDLDescription(x) \rightarrow \exists y.component\_of(x,y) \wedge Parameter(y)$
...
$\forall x.Object(x) \rightarrow \exists y.played\_by(x,y) \wedge Software\_Module(y)$
$\forall x.Operation(x) \rightarrow \exists y.played\_by(x,y) \wedge Method(y)$
$\forall x.Argument(x) \rightarrow \exists y.played\_by(x,y) \wedge Formal\_Parameter(y)$
...

9.5 Example

Last but not least, the example in Figure 8 shows both an APIDescription and an IDLDescription of a KAON Ontology Store which is part of the KAON Tool suite [BEH+02]. For the sake of brevity, we limit ourselves to one Method 'AddStatement' which is part of the KAONOntologyStore Software Module and fulfills the task of storing a triple.

In our context, the KAONOntologyStore**API**Description plays the role of a functional component deployed to the Application Server. The description features several parameters, such as representationLanguage and the ComponentID. Furthermore, the Functional Component has exploitation within the StoreTriple task.

The KAONOntology**IDL**Description consists only of roles: Object is played by the KAONOntologyStore Software Module, Operation is played by the AddStatement Method, Argument played by a Formal Parameter and so on.

Note that an APIDescription is expected to have several Tasks, like StoreTriple, Query, Retrieve and so on. The same holds for IDLDescription which should feature one Object role related to a multitude of Operation roles.

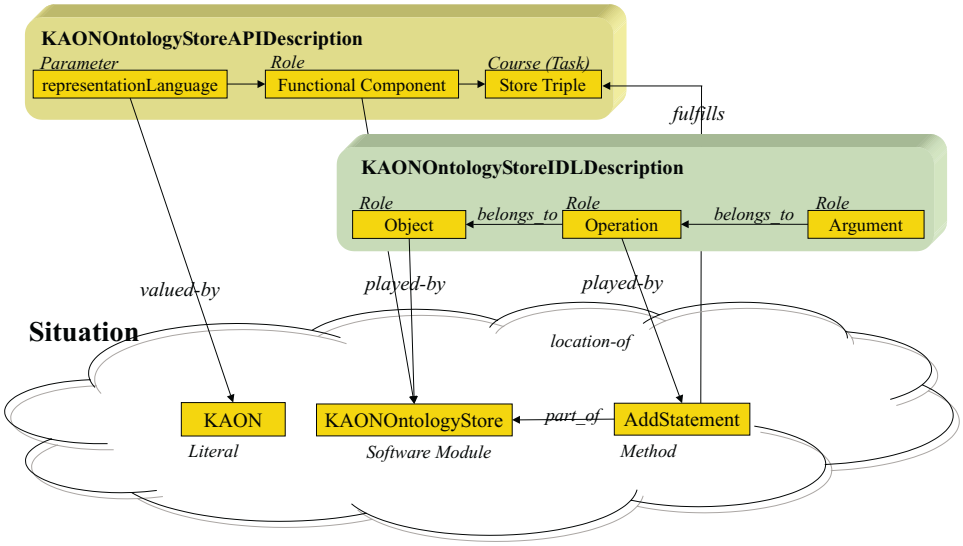

Figure 8: KAON Ontology Store Example

References

[ABFG02] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. Web services architecture requirements. http://www.w3.org/TR/wsa-reqs, Nov 2002.

[BCF+03] David Booth, Michael Champion, Chris Ferris, Francis McCabe, Eric Newcomer, and David Orchard. Web Services Architecture, May 2003.

[BEH+02] E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer, G. Stumme, Y. Sure, J. Tane, R. Volz, and V. Zacharias. KAON - towards a large scale Semantic Web. In Kurt Bauknecht, A. Min Tjoa, and Gerald Quirchmayr, editors, *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*, volume 2455 of *Lecture Notes in Computer Science*. Springer, 2002.

[CCMW03] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL). `http://www.w3.org/TR/wsdl`, Mar 2003. W3C Note.

[Coa03] The DAML Services Coalition. DAML Services, May 2003.

[Fel98] Christiane Fellbaum, editor. *WordNet - An electronic lexical database*. MIT Press, 1998.

[GM] Aldo Gangemi and Peter Mika. Understanding the Semantic Web through Descriptions and Situations. Submitted to ODBASE 2003.

[GM03] Aldo Gangemi and Peter Mika. Understanding the semantic web through descriptions and situations. In *DOA/CoopIS/ODBASE 2003 Confederated International Conferences DOA, CoopIS and ODBASE, Proceedings*, LNCS. Springer, 2003.

[GPS99] Aldo Gangemi, Domenico M. Pisanelli, and Geri Steve. An overview of the ONIONS project: Applying ontologies to the integration of medical terminologies. *Data Knowledge Engineering*, 31(2):183–220, 1999.

[Gro02] Object Modelling Group. Idl / language mapping specification - java to idl, Aug 2002. 1.2.

[Köh29] Wolfgang Köhler. *Gestalt Psychology*. Liveright, New York, 1947/1929.

[MBD+03] David Martin, Mark Burstein, Grit Denker, Jerry Hobbs, Lalana Kagal, Ora Lassila, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Marta Sabou, Evren Sirin, Monika Solanki, Naveen Srinivasan, and Katia Sycara. DAML-S (and OWL-S) 0.9 draft release. `http://www.daml.org/services/daml-s/0.9/`, Nov 2003.

[MBG+02] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, Alessandro Oltramari, and Luc Schneider. The WonderWeb Library of Foundational Ontologies. WonderWeb Deliverable 17, 2002.

[Mil02] Alexander Miller. *The Stanford Encyclopedia of Philosophy*, chapter Realism. Stanford University, winter edition edition, 2002.

[Moo02] Michael S. Moore. Legal Reality: A Naturalist Approach to Legal Ontology. *Law and Philosophy*, 21(6):619–705, 2002.

[OSRV03] D. Oberle, M. Sabou, D. Richards, and R. Volz. An ontology for semantic middleware: extending DAML-S beyond web-services. In *On the Move to Meaningful Internet Systems and Ubiquitous Computing, 2003 - DOA/CoopIS/ODBASE 2003 Confederated International Conferences DOA, CoopIS and ODBASE 2003 Catania, Sicily, Italy, November 3 - 7, 2003, Workshops*, Lecture Notes in Computer Science. Springer, 2003. In press.

[OVMS03] D. Oberle, R. Volz, B. Motik, and S. Staab. An extensible open software environment. International Handbooks on Information Systems, chapter III, pages 311–333. Springer, 2003.

[Per02] Asuncion Gomez Perez. A survey on ontology tools. OntoWeb Deliverable 1.3, May 2002. www.ontoweb.org.

[vEA02] Ludger van Elst and Andreas Abecker. Ontologies for information management: balancing formality, stability, and sharing scope. *Expert Systems with Applications*, 23(4):357–366, November 2002.

[won] The WonderWeb project (under EU-IST contract IST-2001-33052). See `http://wonderweb.semanticweb.org`.

Appendix

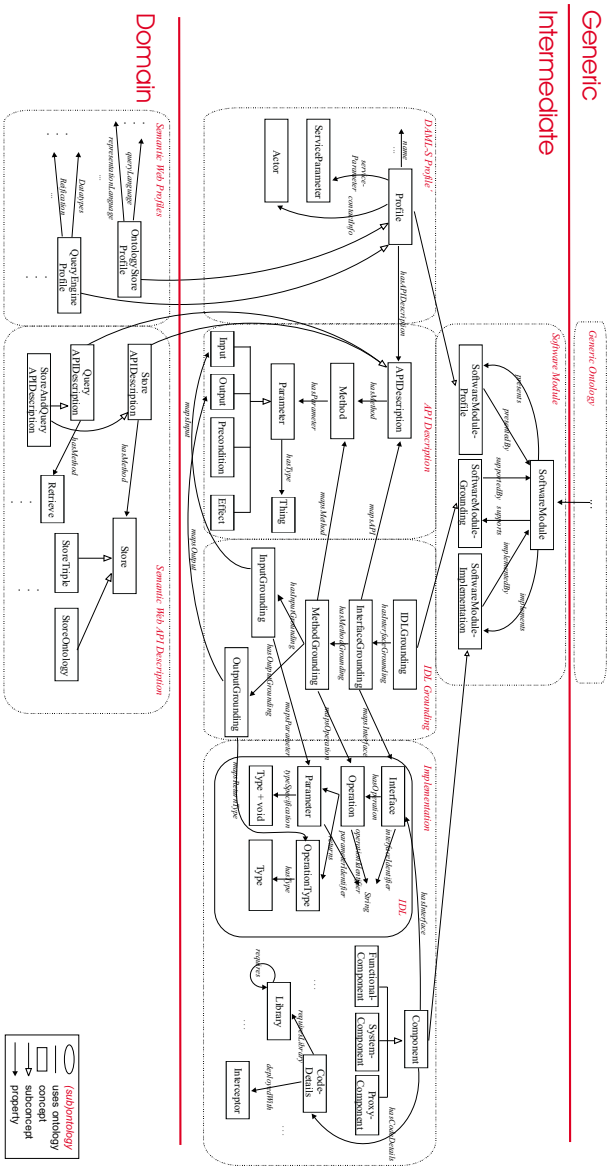

Figure 9: Application Server ontology overview before alignment