WEB SERVICE CAPABILITY META MODEL

Sami Bhiri, Wassim Derguech and Maciej Zaremba
Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland

Keywords: Capability Modelling, Web Services, Business Process, RDF.

Abstract: The concept of capability is a cornerstone element in service description. Nevertheless, despite its fundamental role little effort has been seen to model service capabilities. Current approaches either fail to consider capabilities as feature-based entities and confuse them with annotated invocation interfaces or fail in modelling capabilities at several abstraction levels and establishing links between them. In particular, they are not able to model and deal with concrete capabilities (i.e., capabilities that reflect real customers' needs). In this paper, we propose a conceptual model as an RDF-schema for describing service capabilities. Our model defines capabilities as an action verb and a set of attributes and their values. It is also able to define capabilities at different levels of abstractions/concreteness and establish links between them. Most importantly, our model enables describing concrete capabilities which directly correspond to consumer needs. Our meta model is based on RDF and makes use of Linked Data to define capability attributes as well as their values.

1 INTRODUCTION

From IT perspective, the concept of service has evolved from the notion of remote invocation interface (such as WSDL) to a more comprehensive entity. In this paper, we share the same vision of OASIS Reference Model[1] and consider a service as an access mechanism to a capability. Within this vision the invocation interface is only one aspect of the whole service description. Another core aspect of a service, in which we are interested in this paper, is the notion of capability which describes what a service can do.

The notion of capability is a fundamental concept not only for SOA (Service Oriented Architecture) but also for enterprise information systems. The ARIS architecture (Scheer and Schneider, 2006) recognizes the importance of the functional perspective in enterprise information systems and considers it as one of its views. The concept of capability is the glue point between services and business processes. A service gives access to a certain capability which can be achieved by a business process. Despite its importance, this concept has not drawn the research community attention as it deserves. Current approaches for capability modeling were in fact part of efforts for describing related concepts such as business processes, service descriptions and search requests.

A capability denotes what an action does either in terms of world effects or returned information[1]. In the literature, we can distinguish three families of approaches that tackled the problem of capability modelling either directly or indirectly. The first family includes semantic Web services models (WSMO (Roman et al., 2006) and OWL-S (Martin et al., 2007)) which model capabilities as Input, Output, Preconditions and Effects (IOPE paradigm). Modelling capabilities following the IOPE paradigm does not allow to capture different levels of abstractions of capabilities, easily, and consequently cannot capture any link between them. In particular, these approaches are not suitable for modelling concrete capabilities especially when they are highly configurable. Moreover, they do not feature in an explicit and easily accessible way domain-specifc features of the capabilities which makes their discovery difficult and heavily dependant on reasoning.

The second family of related efforts concern semantic annotations of invocation/interaction interfaces (SA-WSDL and SA-REST) (Kopecký et al., 2007b; Lathem et al., 2007). While these approaches do not directly target capability modelling, they attempt to provide alternative solutions to top-down semantic approaches (WSMO and OWL-S) by starting from existing descriptions such as WSDL and annotate them with semantic information. These ap-

[1]OASIS Reference Model for Service Oriented Architecture 1.0, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

Bhiri S., Derguech W. and Zaremba M..
WEB SERVICE CAPABILITY META MODEL.
DOI: 10.5220/0003935800470057
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 47-57
ISBN: 978-989-8565-08-2

proaches define a semantic (business) description of syntactic interaction interfaces.

The third family includes frame-based approaches for modelling capabilities. Oaks et al., (Oaks et al., 2003) give a nice overview of related approaches and propose a model for describing service capabilities following the same principle. The proposed model distinguishes in particular the corresponding action verb and informational attributes (called roles in the paper (Oaks et al., 2003)) in addition to the classical IOPE. While this model makes a step beyond the classical IOPE paradigm, the semantics of capabilities remain defined via the IOPE paradigm and therefore has the same problems as the first family of approaches described above.

In this paper, we propose an alternative approach, to the classical IOPE paradigm, for modelling capabilities. The semantics of a capability is not captured at a fine-grain level via preconditions and effects. It is rather captured at a coarse-grain level via a combination of (shared meaning of) an action verb, semantic links to other capabilities (which can be defined in an ontology) and, particularly, a set of domain-specific attributes which reflect the business features of the capability. Interestingly enough, our model enables to capture capabilities at different levels of abstraction and establish links between them. Concrete capability descriptions are generated dynamically from more abstract capability descriptions.

The remainder of the paper is organized as follows. Section 2 motivates our work. Section 3 presents our conceptual model for describing capabilities, introduces our attribute-featured principle and details the different types of attributes we consider. It shows also how abstract capabilities can be defined while respecting the attribute-featured principle and serving as declarative specifications for generating concrete capabilities. Section 4 shows how we distinguish different levels of abstractions and shows the relation that may exist between abstract and concrete capabilities and presents semantic links that may exist between them. Section 5 illustrates the relation between the customer request and the capability categories and offers by sketching a matching scenario between the customer request and the capability categories. Section 6 discusses some related work and Section 7 concludes our paper.

2 MOTIVATION

A capability corresponds to what an action (a program, a task, a Business Process etc.) does from a functional perspective. There are different levels of abstraction/concreteness of capabilitiess. A capability can be as abstract as a category and denotes a class of actions, as it can be very concrete and corresponds to a specific consumer request. For instance the category "Shipping" that denotes all functions for shipping is a capability. "Shipping a package of 10 Kg from Ireland to Singapore on the 15^{th} of January 2012 for the price of 200 Dollars" is also a capability but less abstract than the previous one.

In real world settings it is quite often very hard, even impossible, to model statically concrete capabilities due to the following reasons:

- Combinatory explosion: at concrete level, one needs to take care of all possible combinations which may lead to a combaniatory explosion of capabilities (services). For example, a shipping service serves only certain routes (combination of the *source* and *destination* attributes). Defining statically the corresponding concrete capabilities means defining a capability for each route.
- Dynamicity: an aspect of a given capability (the value of an attribute) may depend on a dynamic variable. For instance, the price of a concrete shipping capability may depend on the exchange rate or on the company workload.
- Sensitivity: for business reasons, providers do not want to reveal the values of certain sensitive attributes as part of the static service description. They rather require to engage in a certain interaction before revealing the concrete value of a sensitive attribute.

In this paper, we propose a solution to the above mentioned problems. We propose a conceptual model, as an RDF-Schema, for describing capabilities. Our meta model defines capabilities as an action verb and a set of domain-specific attributes. It is also able to describe capabilities at different levels of abstractions/concreteness and establish links between them. Most importantly, our model enables describing concrete capabilities which are directly consumable by and delivered to consumers contrary to current meta models which describe the abstractions of these descriptions. Our model enables taking into account attributes dynamicity, sensitivity and interdependency. Our model is based on RDF and makes use of Linked Data to define capabilities attributes as well as their values.

Throughout the whole paper, we will consider the shipping domain for illustrating several parts of our conceptual model. In such domain, from the service consumer perspective, it is much more valuable to operate on concrete, consumable shipping offers such "*Shipping using FedEx Europe First, price 100*

EUR, from Germany to Ireland, delivery within 1 day" rather than to operate on abstract service descriptions such "*FedEx shipping company ships packages up to 68 Kg, between various source and target destinations*". Current service descriptions require manual work in order to move to the service offer level. Our conceptual model aims at making this step automated.

3 CAPABILITY META MODEL

In this Section, we introduce the concept of capability, then we explain our attribute-featured principle that we consider for modelling the capability attributes and we present the possible attribute types covered by our meta-model.

Definition 1 introduces the concept of capability in our meta-model.

Definition 1. (Capability) A tuple *Cap = (ActionVerb,Attributes)* is a capability, where:

- *ActionVerb*: This concept has been previously introduced by (Oaks et al., 2003) in order to define, in a natural language, what is the action being described. Different to (Oaks et al., 2003), we consider the action verb as a concept from a domain related ontology that comes form a shared aggreeement on its semantics.
- *Attributes*: Represents a set of pairs *(Attribute, AttributeValue)* that corresponds to the set of characteristics of the capability. An *Attribute* corresponds to a particular property of the capability and *AttributeValue* corresponds either to the value or the possible values that this *Attribute* can have.

Figure 1 shows a detailed UML class diagram of the capability metal model. It depicts the concepts previously introduced (i.e., Capability, ActionVerb, Attribute and AttributeValue) as well as other ones that will be introduced in the rest of this section.

Additionally, in our work, we distinguish between several abstraction levels of capabilities. As shown in Figure 1, we consider the relations "*specify*" and "*extend*" to express the relations between capabilities. These relations will be detailed later in this paper.

In the rest of this section, we will discuss our attribute-featured principle in Section 3.1 as well as the possible AttributeValue types in Section 3.2. Section 4, will define the capability abstraction relations.

3.1 Attribute-featured Principle

We have, recently, noticed a wide adoption of the Linked Data (Bizer et al., 2009; Sheridan and Tennison, 2010) principles, and a growing amount of data sets specified in RDF. It is clear that Linked Data provides the best practices for publishing structured data on the Web. Linked Data is published using RDF where URIs are the means for referring various entities on the Web giving the possibility to interlink them. Currently, organizations are highly interested in publishing their data in RDF (Lebo and Williams, 2010) as well as various public vocabularies (ontologies) are being released. Consequently, we have decided to define our meta model based on RDF and make use of Linked Data principles in order to define capability attributes as well as their values. Listing 1 shows a fragment of the capability Ontology in RDF using a human readable N3 notation.

In Listing 1, we define the concept *cap:Capability* as an *rdfs:Class*. We define the concept *cap:ActionVerb* as a sub class of *skos:Concept*. "*The skos:Concept class allows to assert that a resource is a conceptual resource. That is, the resource is itself a concept.*"[2]. And we define the concept *cap:AttributeValue* as an equivalent class to *owl:Thing* because we need it to be the most general class to allow for the reuse of existing vocabularies for possible attribute values for example using *vcard* open vocabulary for defining addresses.

Listing 1: Capability ontology snippet.

```
@prefix rdf: <http://www.w3.org/1999/02/
             22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl
    #>.
@prefix rdfs: <http://www.w3.org/2000/01/
              rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos
    /
              core#>.
@prefix cap: <http://.../ontology/capability
    #>.

cap:Capability a rdfs:Class.

cap:ActionVerb rdfs:subClassOf skos:Concept.

cap:AttributeValue owl:equivalentClass owl:
    Thing.

cap:do a rdf:Property;
   rdfs:domain cap:Capability;
   rdfs:range cap:ActionVerb.

cap:attribute a rdf:Property;
   rdfs:domain cap:Capability;
   rdfs:range cap:AttributeValue.
```

[2]SKOS Core Guide, W3C Working Draft 2 November 2005, http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/

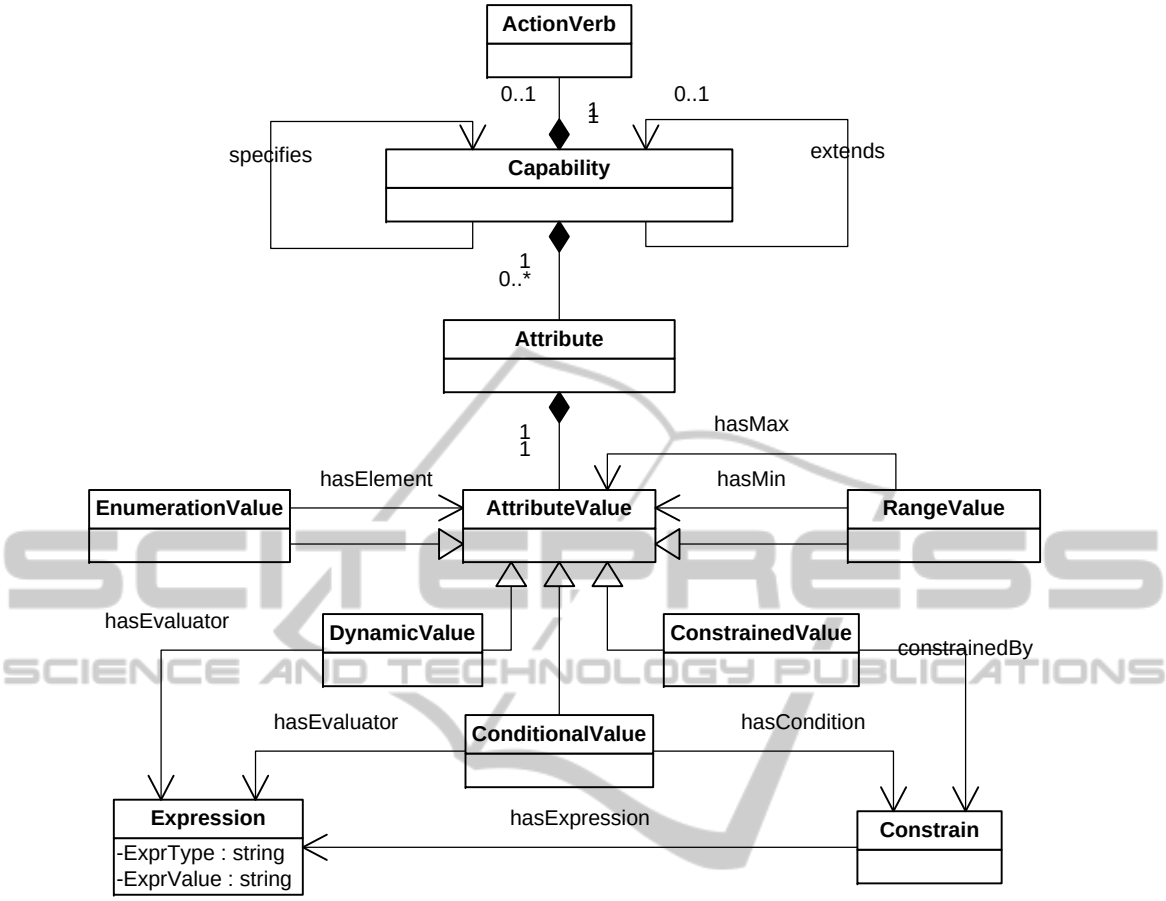

Figure 1: Capability UML class diagram.

The property *cap:do* allows linking a *cap:Capability* to its corresponding *cap:ActionVerb*. We created the property *cap:attribute* even though we are going to redefine it for each created attribute its corresponding range. This will have the advantage to create properties where the domain is always set to a *cap:Capability*. In other words, all attributes that will be created as an *rdfs:subProperty* of *cap:attribute* will be interpreted as a property of a capability.

We can define a domain related ontology in order to define a particular capability (i.e., its action verb and attributes). Taking as example the shipping domain, Listing 2 shows a snippet of such ontology. In order to define the action verb, we use *skos:prefLabel* (Line 6). The rest of the listing illustrates how two attributes *cap:from* and *cap:pickUpDate* are defined in our shipping ontology. Both attributes are *rdfs:subProperty* of *cap:attribute* (Line 8 and 11).

The range of the attribute *cap:from* is *vcard:VCard* to refer to an address. The range of the attribute *cap:pickUpDate* is a concept defined in the same shipping ontology. Other attributes in subsequent listings are defined similarly.

Listing 2: Shipping domain ontology snippet.

```
1  @prefix vcard: <http://www.w3.org/2006/vcard
       /ns#>.
2  @prefix cap:   <http://.../ontology/
       capability#>.
3  @prefix ship:  <http://.../ontology/
       ship_domain#>.
4
5  :shipment a cap:ActionVerb;
6            skos:prefLabel ''Ship''.
7
8  ship:from rdfs:subProperty cap:attribute;
9            rdfs:range vcard:VCard.
10
11 ship:pickUpDate a rdfs:subProperty cap:
       attribute;
12                 rdfs:range ship:date.
```

3.2 Attribute Value Types

This section details how the set of attributes of a certain capability can be defined from a service provider perspective. As depicted in Figure 1, we consider five classes used for describing the values of a capability

attributes.

Using these classes separately or in combination, a capability can specify (i) the possible values attributes can have, (ii) and how to compute their values. We use a capability, called *BlueCap*, as an example to illustrate the use of these classes. Before detailing these classes, we need to introduce the concepts of *Constraint* and *Expression* which some attribute values may refer to.

3.2.1 Constraint and Expression

A constraint enables to specify the possible values an attribute can have. The class *Constraint* represents all constraints. The class *Expression* enables to specify expressions among them the value of a given constraint. The class *Expression* has two attributes/properties, *ExprType* which specifies the type of the expression and *ExprValue* which defines the expression itself. The type of the expression, *ExprType*, indicates how to build the corresponding queries during a matching process. Currently, the only type of expression our meta model supports is SPARQL (queries).

Listing 3 shows an example for expressing a constraint on the weight of the package. The constraint *PackgConstraint* is defined in line 1. This constraint has an expression of type SPARQL. The value of the constraint expression (line 6) indicates that the weight of the package has to be lower than or equal to 50 Kg.

Listing 3: Example of a constraint.

```
1  :PckgConstraint      a              cap:
       Constraint;
2          cap:hasExpression    :
               PckConstraintExpr.
3
4  :PckgConstraintExpr  a              cap:
       Expression;
5                       cap:exprType   "SPARQL";
6                       cap:exprValue  "?weight
                            =< 50?
7                       && ?weightUnit =
                            dbpedia:KG".
```

3.2.2 Constrained Value

The class *ConstrainedValue* enables defining the possible values an attribute can have by specifying a set of constraints on its value. As depicted in Figure 1, a *ConstrainedValue* is constrained by a set of constraints. Listing 4 shows how *BlueCap* can specify that it can ship packages of weight under 50 Kg. The value *X*, of the attribute *Item*, is a *ConstrainedValue* (lines 1 and 3). X is constrained by the constraint *PckgConstraint* (line 6) which is detailed in Listing 3.

Listing 4: Example of a constrained value.

```
1  :BlueCap        ship:Item              :X.
2
3  :X      a  ship:Package, cap:
       ConstrainedValue;
4          ship:hasWeight  [ship:hasValue ?
               weight;
5                          ship:hasUnit ?
                               weightUnit];
6          cap:constrainedBy       :
               PackgConstraint.
```

3.2.3 Dynamic Value

A *DynamicValue* defines how to compute the value of an attribute which value depends on (i) consumer provided attributes, (ii) dynamic values, or (iii) hidden variables. As shown in Figure 1 a *DynamicValue* refers to an expression that defines how to compute it.

Listing 5 shows an example of how to compute the shipping price. The value *Y*, of the attribute *price*, is a *DynamicValue*. It has as evaluator the expression *PriceExpression* (lines 6) which is a SPARQL expression (line 9). Line 10 specifies the formula for computing the price based on the weight of the package.

Listing 5: Example of a dynamic value.

```
1  :BlueCap        ship:price             :Y.
2
3  :Y      a  ship:ShippingPrice, cap:
       DynamicValue;
4          ship:hasValue    ?price;
5          ship:hasUnit     dbpedia:USD;
6          cap:hasEvaluator :PriceExpression.
7
8  :PriceExpression a cap:Expression;
9                   cap:hasType    "SPARQL";
10                  cap:exprValue  "?price :=
11                  fn:ceiling(?weight)
                        *5.5+41".
```

3.2.4 Conditional Value

A *ConditionalValue* assigns an Attribute Value to the corresponding attribute if a certain condition holds. As shown in Figure 1, a *ConditionalValue* has a condition expressed as a constraint and an element which corresponds to the corresponding attribute value.

Listing 6 gives an example showing how to specify a shipping price when the target country is a European country. The value Y, of the attribute *price*, is a *ConditionalValue* (lines 3). It assigns the Attribute Value, *EuropeanPrice* (line 6), when the *PriceCondition* holds (line 5). *PriceCondition* is a *Constraint* which requires that the target country is in Europe (Lines 8-12). *EuropeanPrice* is a *DynamicValue* (line

14) and has as evaluation expression *PriceExpression* (line 15) which is detailed in Listing 5.

Listing 6: Example of a conditional value.

```
1  :BlueCap        ship:price              :Y.
2
3  :Y      a ship:ShippingPrice, cap:
       ConditionalValue;
4          ship:hasValue          ?price;
5          cap:hasCondition       :
               PriceCondition;
6          cap:hasElement         :EuropeanPrice
               .
7
8  :PriceCondition a cap:Constraint;
9        cap:hasExpression  [cap:hasType "
             SPARQL";
10                          cap:hasValue "?
                                trgCountry
11                          skos:subject
                                dbpedia-
12                          cat:European
                                _countries"].
13
14 :EuropeanPrice a   cap:DynamicValue;
15                  cap:hasEvaluator   :
                        PriceExpression.
```

4 DIFFERENT LEVELS OF ABSTRACTIONS

In our meta model the semantic of a capability derives not only from the Action Verb concept but also from its relations (semantic links) with other capabilities. These relations are established based on capabilities attributes for some, and on their attribute values for others.

As mentioned above, capabilities can be defined at several abstraction levels. A capability that is proposed to an end user is actually the least abstract (i.e., a concrete) capability that we call *Capability Offer*. In contrast, each abstract capability is called *Capability Category*. Navigation between different abstraction levels is made through semantic links between the various capabilities.

In this paper, we present three special relations: `variantOf`, `specifies` and `extends`, which enable defining a hierarchy of capabilities in a given domain, from the most general and abstract one to the most concrete ones. Definitions 2, 3 and 4 define respectively the relations `variantOf`, `specify` and `extend`.

Please note:

- For abbreviation purposes and by misnomer we say that a certain capability *cap* has an attribute *at*.
- We refer to the attribute *at* of *cap* by *cap.at*.
- We refer to the set of attributes of *cap* by *cap.attributes*.
- We say that two capabilities cap_1 and cap_2 (or more) share the same attribute *at* if both of them have the attribute *at* (but possibly with a different value).

Definition 2. (variantOf) this relation holds between two resources (A capability is itself a resource.). It unifies, in fact, the two properties *rdf:type* and *rdfs:subClassOf*[3]. A resource r_1 `variantOf` a resource r_2 if r_1 is an instance (*rdf:type*) or a subclass (*rdfs:subClassOf*) of r_2.

Definition 3. (specifies) Given two capabilities cap_1 and cap_2, cap_1 `specifies` cap_2 if (i) all the attributes of cap_2 are also attributes of cap_1 (In other terms cap_1 inherits all the attributes defined in cap_2), (ii) for every shared attribute *at*, the value of $cap_1.at$ is either equal to or `variantOf` the value of $cap_2.at$, and (iii) there exists at least one shared attribute *at'* such that the value of $cap_1.at'$ `variantOf` $cap_2.at'$.

Definition 4. (extends) Given two capabilitites cap_1 and cap_2, cap_1 extends cap_2 if (i) cap_1 has all the attributes of cap_2 and has additional attributes, and (ii) for every shared attribute *at*, the value of $cap_1.at$ is equal to the value of $cap_2.at$.

The relations `specifies`, `extends` enable defining a hierarchy of capabilities. In Figure 2, we show an example of a hierarchy of capability descriptions. As a root of this hierarchy, we define *Capability A*. It represents an abstract capability description for shipping goods from any source and destination at an international scale. This capability can be extended either to *Capability B* or *Capability C*. Both extend the initial capability by 1 or 2 attributes. As an example of specification relation between capabilities, we refer to the link between *Capability D* and *Capability B*. *Capability D* specifies *Capability B* as it becomes a European shipping capability instead of International. It is also quite clear that *Capability E* extends *Capability D*.

Similar to domain ontologies which define shared concepts and shared attributes/properties, top level capability in a given domain can also be defined as an ontology where an agreement about their meaning is reached and shared. Like any other ontology concepts, this capability top level layer can be reused to define other capabilities.

While this coarse-grain semantics is adequate for conducting discovery, it is insufficient when auto-

[3]We consider instantiation a special kind of sub classing where the instance is in fact no more than a sub class (subset) with only one element.

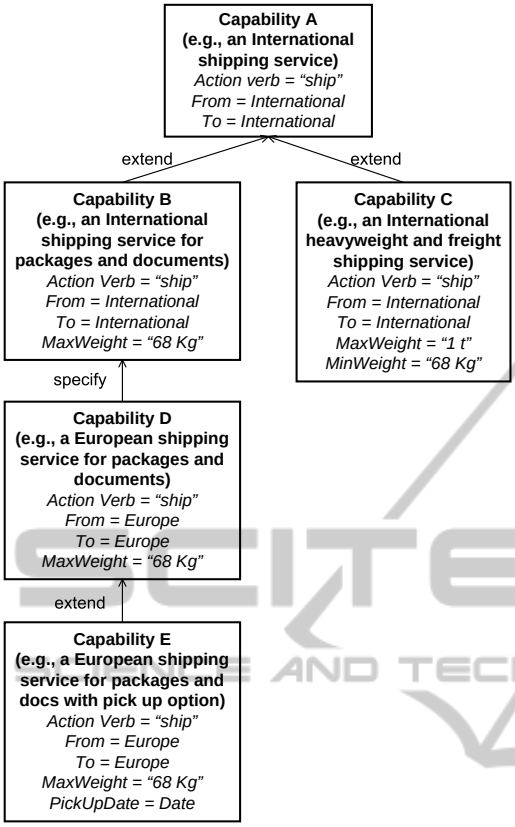

Figure 2: Example of shipping capabilitities hierarchy.

mated chaining is required such as in composition and planning scenarios. By coarse grain semantics we mean defining shared agreement on coarse grain entities such as capability in our case. Automated chaining requires finer-grained semantics. For that purpose, we consider two additional attributes *cap:before* and *cap:after*. That we are not discussing in detail in this paper. But simply put, taken together these two attributes capture the change to the state of the world that the corresponding action performs.

5 CUSTOMER REQUEST, CAPABILITY CATEGORY AND CAPABILITY OFFER

An important concept, complementary to the concept of capability in many scenarios, is the concept of *Consumer Request* which specifies what the client wants to achieve. This concept deserves a conceptual model for its own. In the following, we need to refer to some parts of the customer request, in an informal way, to show the interplay between it and capabilities. But we do not elaborate beyond that informal description.

In order to illustrate the relation between capability category, offer and customer request, we give an overview of the matching process between a customer request and a capability category. The purpose is not to cover in detail the matching or discovery phase.

Let's consider for our case a shipping company called *Blue* that has a special shipping offer called *BlueOffer*. Listing 7 illustrates how we can define the capability *BlueOffer* according to our capability model. *BlueOffer* consists of offering: "*a shipping service from Ireland to Singapore of a package of 10 Kilograms. The shipping will cost 200 USD and the package will be delivered the earliest on the 12th of December 2011.*"

The description of concrete capabilities such as *BlueOffer* (See Listing 7) depends on the corresponding customer request. More specifically the values of certain attributes are in fact defined in the customer request. For instance the values of the attributes *ship:from* and *ship:to* of the capability *BlueOffer* (described in Listing 7) are specified in the given customer request.

Before going into detail about the matching phase (See Section 5.2), we need to introduce the attribute types based on who is providing the value of that attribute in Section 5.1.

Listing 7: BlueOffer capability description snippet.

```
:blueOffer a  cap:Capability;

           cap:do [a cap:ActionVerb;
                   skos:prefLabel ''Ship''];

           ship:from [a ship:SourceAddress;
                      vcard:address dbpedia:
                          Ireland];

           ship:to [a ship:TargetAddress;
                    vcard:address dbpedia:
                        Singapore];

           ship:item [a ship:Package;
                      ship:hasWeight [ship:
                          hasValue 10;
                      ship:hasUnit dbpedia:
                          Kilogram]];

           ship:price [a ship:ShippingPrice;
                       ship:hasValue 200;
                       ship:hasUnit dbpedia:
                           USD];

   ship:deliveryDate [a ship:
       ShippingDeliveryDate;
                      ship:earliest
                          ''2011-12-13''
                      ^^xsd:date].
```

5.1 Attribute Types

We distinguish three types of attributes depending on the source of their values. In general, three sources are involved in a scenario of capability consumption. These sources are the *consumer* who has a request and is looking for a (concrete) capability to achieve, (ii) the *provider* who provides capabilities for achieving consumer requests and (iii) the *interaction context*.

The values of a concrete capability attributes are provided/computed by one of these players. According to who provides the value of an attribute, we distinguish three subclasses of attributes namely *Co*, *Pro* and *Co&Pro* attributes. It is important to emphasize that this classification is *based on which role provides the value of the corresponding attribute*:

- *Consumer or Context* (**Co**) attributes are attributes which values are specified by the consumer or provided by the interaction context. For instance the attributes *ship:to* and *ship:from* in the *BlueOffer* capability are **Co** attributes.
- *Provider* (**Pro**) attributes are attributes which values are specified or computed by the capability provider. For instance the attribute *ship:Price* in the *BlueOffer* capability is a **Pro** attribute.
- *Consumer then Provider* (**Co&Pro**) attributes are attributes which values are firstly specified by the consumer but may be changed by the provider. For instance, the attribute *ship:DeliveryDate* is a **Co&Pro** attribute; the consumer can select his preferred delivery date and the provider can change it if some of his working constraints do not fit the exact proposed date.

5.2 Matching

By analogy to Object Oriented Programming, a capability category can bee seen as a class and capability offers as objects. Similar to objects, which are instantiated from classes, capability offers are generated from capability categories. Certain capability categories are, declarative specifications for generating capability offers for particular consumer requests. If possible, a capability offer is dynamically generated from a capability category for a particular consumer request. We say that a particular capability offer is variant of the capability category it derives from.

Figure 3 gives an overview of the matching process and shows which artifact is used in each step. The matching phase has as input the customer request and a capability category and as output the customer offers variant that matches the request.

As shown in Figure 3, a consumer request specifies (i) the values of Co attributes, (ii) hard constraints on some Pro and Co&Pro attributes, and (iii) preferences on Co, Pro and Co&Pro attributes. A capability category specifies (i) the possible values Co and Co&Pro attributes can have while considering their interdependencies, and (ii) the values or how to compute the values of Pro and Co&Pro attributes.

The matching process consists of:

- **The First Step** checks whether the capability category supports the values of the Co and Co&Pro attributes specified in the customer request. If the answer is negative then there is no capability offer variant of the given category that matches the customer request. If the answer is positive, the process moves to the following step.
- **The Second Step** consists of generating the capability offers that correspond to the customer request. A capability category and its offers have the same attributes but with different values for some of them. The generated capability offers will be constructed as follows. The value of Co attributes will be those specified by the customer request. The value of the Pro and Co&Pro attributes will be specified or computed as described in the capability category.
- **The Third Step** consists of checking which of the generated offers satisfy the hard constraints specified by the customer. Those capability offers which do, will be maintained as candidates. The matching phase should be followed by a ranking phase where the capability offer candidates will be ranked based on the consumer preferences.

6 RELATED WORK

Even though the concept of capability has a central role in enterprise information systems, it has not got much attention and few works have been interested in modelling capabilities as such.

The Semantic Web uses ontologies and languages allowing several ways to describe web services. For example WSDL-S[4] and its successor SA-WSDL (Kopecký et al., 2007b; Lathem et al., 2007) use ontologies to enrich WSDL description and XML Schema of Web services with semantic annotations. Such techniques consider a capability as an invocation interface. However, as explained in this paper a capability is not an interface. It is an entity featured via a set of attributes. We believe that our vision and understanding of capability is more accurate. The background of the genesis of SOA, which is distributed method invocation, has influenced these techniques.

[4]http://www.w3.org/Submission/WSDL-S.

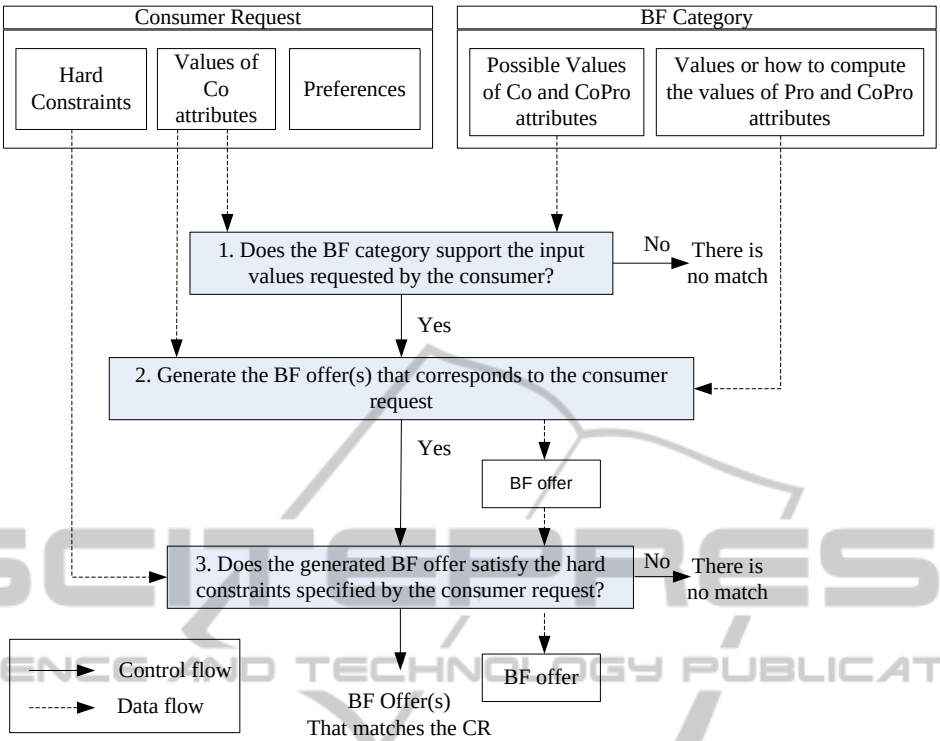

Figure 3: Overview of the matching process of a customer request and a capability.

In a more refined fashion, languages such as OWL-S (Martin et al., 2007), WSMO (Roman et al., 2006), SWSO[5], provide a semantic description of Web services. However, these approaches do not go beyond the classical Input, Output, Precondition and Effect paradigm to define services capabilities. They do not feature the business aspects of a capability. In addition, they describe capabilities at an abstract level. They are not able to model concrete capabilities that correspond to specific needs of consumers. However, what clients are interested in are concrete capabilities. The matching of consumer requests has to be against concrete capabilities.

Oaks et al., (Oaks et al., 2003) propose a model for describing service capabilities going one step beyinf the IOPE paradigm by distinguishing in particular the corresponding action verb and informational attributes (called roles in the paper (Oaks et al., 2003)). However, the semantics of capabilities remain defined via the IOPE paraddigm and therefore has the same problems of the previously described approaches.

The notion of (service) offer is currently not addressed in service description languages such as Universal Service Description Language(USDL[6]). Service descriptions have a varying level of concreteness depending on the stage of a matchmaking and configuration. This notion is currently missing from the USDL. Introducing different levels of concreteness in USDL Functional Module should make USDL applicable for describing highly configurable services.

Other related works worth mentioning are (Kopecký et al., 2007a; Vitvar et al., 2007; Zaremba et al., 2006). These works have identified the gap between current modeling techniques and real world requirements and initiated the discussions about abstract services and concrete offers descriptions. Similar to all related work, the concept of capability was not tackled as first-class entity. The focus was rather on the service model. In addition, (Vitvar et al., 2007) and (Zaremba et al., 2006) rely on and extend the Input, Output, Precondition and Effect paradigm without making explicit and clear the business features of services functionalities. They also do not explicitly distinguish between abstract services and service offer, nor do they define the links between them.

7 CONCLUSIONS

In this paper, we presented an original meta model for describing Capabilities which presents several advantages.

[5]http://www.w3.org/Submission/SWSF-SWSO
[6]www.internet-of-services.de

First, contrary to the Input, Output, Precondition and Effect paradigm it features the business and functional characteristics which consumers are mostly interested in and which are specified in their requests.

Second, our meta model can deal with capabilities at different abstraction levels in a uniform way. In addition, it establishes relations between Capabilities at different abstraction levels. In particular, it provides the required declarative specification to dynamically generate concrete Capabilities from abstract ones.

Furthermore, our meta model defines semantic links between Capabilities. By using these relations Capability owners can rapidly and easily define new Capabilities by reusing previous definitions. In addition, these relations define a cloud of Capabilities where navigation techniques can be developed as an alternative to goal based discovery techniques.

A cloud of capabilities description can be easily queried using SPARQL. Actually, we use RDF as a lightweight language for describing and linking capabilities descriptions whereas we can use SPARQL for advanced querying including the usage of SPARQL as a rule language (Axel Polleres, 2007).

Finally, since our meta model is RDF based it can be easily extended, while preserving the attribute-featured principle, by considering other types of attributes (such as optional and mandatory attributes) and other types of attribute values.

We did not give much attention in this paper to the concept of Action Verb. We do not want to tight the use of this concept to a simple verb. It can be an expression describing the actual service action. It is also possible to enrich this concept by other related terms such as synonyms. Natural Language Processing (Sugumaran and Storey, 2003) can be applied together with WordNet verb synsets for generating possible synonyms to enrich the action verb concept of the capability description.

It is possible to use concepts from a domain related ontology/taxonomy or a categorization schema like NAICS[7] or UNSPSC[8] for this Action Verb in order to be more compliant to the domain vocabulary.

Otherwise, as it has been stated by (Oaks et al., 2003), we can use the MIT Process Handbook (Malone et al., 2003) for determining the action verb of a service capability. A capability can be matched to a particular process in the process handbook. We can also use the meronymy and the hyponymy relations described in that process handbook for creating the hierarchy of capabilities.

[7]North American Industry Classification System, http://www.census.gov/eos/www/naics/

[8]United Nations Standard Products and Services Code, http://www.unspsc.org/

As part of our future work, we plan to:

- investigate other relations that might be useful for creating the capabilities hierachy/cloud. The possible other relations under definition are: *share*, *shareSame*, *shareDifferent*, *differMore* and *differLess*. Some of these relations do not bring much information on themselves. They are used in fact to compute other relations.
- explore what we call extended relations in contrast to basic relations that we are currently considerering. Basic relations are coarse-grain relations/links while extended relations are more fine-grained. An extended relation specifies which attribute the basic relation, it derives from.
- provide an automation support for maintaining the capabilities hierarchy. We aim to provide an automation support to add or remove any capability from the hierarchy/cloud.

ACKNOWLEDGEMENTS

This work is funded by the Lion II project supported by Science Foundation Ireland under grant number 08/CE/I1380.

REFERENCES

Axel Polleres (2007). From SPARQL to rules (and back). In *WWW 2007, Banff, Alberta, Canada*. ACM.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked Data - The Story So Far. *IJSWIS*, 5(3).

Kopecký, J., Simperl, E. P. B., and Fensel, D. (2007a). Semantic Web Service Offer Discovery. In *SMRR*, volume 243 of *CEUR Workshop Proceedings*.

Kopecký, J., Vitvar, T., Bournez, C., and Farrell, J. (2007b). Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6).

Lathem, J., Gomadam, K., and Sheth, A. P. (2007). Sa-rest and (s)mashups : Adding semantics to restful services. In *ICSC*. IEEE Computer Society.

Lebo, T. and Williams, G. T. (2010). Converting governmental datasets into linked data. In *I-SEMANTICS*. ACM.

Malone, T. W., Crowston, K., and Herman, G. A., editors (2003). *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, 1st edition.

Martin, D., Paolucci, M., and Wagner, M. (2007). Bringing semantic annotations to web services: Owl-s from the sawsdl perspective. In *ISWC/ASWC*, volume 4825 of *LNCS*. Springer.

Oaks, P., ter Hofstede, A. H. M., and Edmond, D. (2003). Capabilities: Describing What Services Can Do. In *ICSOC*, volume 2910 of *LNCS*. Springer.

Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Domingue, J., Bussler, C., and Fensel, D. (2006). Www: Wsmo, wsml, and wsmx in a nutshell. In *ASWC*, volume 4185 of *LNCS*. Springer.

Scheer, A.-W. and Schneider, K. (2006). ARIS Architecture of Integrated Information Systems. *Bernus Peter Mertins Kai Schmidt Gunter*.

Sheridan, J. and Tennison, J. (2010). Linking UK Government Data. In *LDOW 2010*.

Sugumaran, V. and Storey, V. C. (2003). A semantic-based Approach to Component Retrieval. *SIGMIS Database*, 34.

Vitvar, T., Zaremba, M., and Moran, M. (2007). Dynamic Service Discovery Through Meta-interactions with Service Providers. In *ESWC*, volume 4519 of *LNCS*. Springer.

Zaremba, M., Vitvar, T., Moran, M., and Haselwanter, T. (2006). WSMX Discovery for the SWS Challenge. In *ISWC*, Athens, Georgia, USA.