Capabilities: Describing What Services Can Do*

Phillipa Oaks, Arthur H.M. ter Hofstede, and David Edmond

Centre for Information Technology Innovation - Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia
{p.oaks,a.terhofstede,d.edmond}@qut.edu.au

Abstract. The ability of agents and services to automatically locate and interact with unknown partners is a goal for both the semantic web and web services. This, "serendipitous interoperability", is hindered by the lack of an explicit means of describing what services (or agents) are able to do, that is, their capabilities. At present, informal descriptions of what services can do are found in "documentation" elements; or they are somehow encoded in operation names and signatures. We show, by reference to existing service examples, how ambiguous and imprecise capability descriptions hamper the attainment of automated interoperability goals in the open, global web environment. In this paper we propose a structured, machine readable description of capabilities, which may help to increase the recall and precision of service discovery mechanisms. Our capability description draws on previous work in capability and process modeling and allows the incorporation of external classification schemes. The capability description is presented as a conceptual meta model. The model supports conceptual queries and can be used as an extension to the DAML-S Service Profile.

1 Introduction

In recent times the Semantic Web, and Web Services have converged into the notion of self-describing semantic web services. These are web services that provide and use semantic descriptions of the concepts in their domain over and above the information provided by WSDL[1] and UDDI[2]. Two W3C groups (Semantic Web and Web Services) have described a need for service descriptions that are sufficiently expressive to allow services to be located dynamically without human intervention. The requirements for the W3C's Web Services Architecture and Web Services Description working groups describe the need for "semantic descriptions that allow the discovery of services that implement the required functionality" [1]. The Web Ontology Language (OWL) requirements describe "serendipitous interoperability" as the ability of devices to "discover each others'

* This work is supported by the Australian Research Council SPIRT Grant "Self-describing transactions operating in a large, open, heterogeneous and distributed environment" involving QUT, UNSW and GBST Holdings Pty Ltd.

[1] http://www.w3.org/TR/wsdl12/

[2] http://www.uddi.org

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 1–16, 2003.

functionality and be able to take advantage of it" [2]. This sentiment can also be applied to web services.

At present web services are described using the Web Services Description Language (WSDL). WSDL only provides for the description of web service interfaces. There are two alternative ways of dealing with this as far as determining the capability of the service is concerned. The first way is for users to manually search for services and read the documentation to see what the service can do, then hard wire the service invocation and interaction. The second way is to locate services based on matching keywords representing the required capability with words used in the interface description. This is particularly prone to problems, as most software developers use names and words idiosyncratically according to their local culture, rules and naming conventions. These conventions often require words to be mashed together, and the words may or may not bear any relevance to what the service actually does. In the world of web services it is no longer possible to assume that all users will share the local conventions of the service provider.

Service descriptions must explicitly state what they can do, and the context the service operates on and within. The advantage of this higher level capability description, is that users can select and use specific functionality. This is in contrast to the current situation, where service users must make their own determination of the functionality and requirements of each operation.

To enable the dynamic discovery of services, a mechanism is required to describe behavioural aspects of services, i.e. what services do. A semantic description of services should include the following: the capabilities a service can provide, under what circumstances the capabilities can be provided, what the service requires to provide the capability, and what results can be delivered. The description should also provide references to definitions of all the words and terms it uses. The capability description has to provide enough information for users to identify and locate alternative services without human intervention. Non-functional aspects of services, such as cost and quality of service [3] are necessary for the evaluation, selection, and configuration of services following their discovery. A description of these aspects is outside of the scope of this work.

In this paper we are concerned with advertising web services in such a way that the discovery of their capabilities can be automated. Although the other phases of service interaction, such as evaluation, selection, negotiation, execution and monitoring are important, the discovery phase is the crucial first step.

2 Existing Work in Capability Description

A set of criteria for evaluating capability description languages were described by Sycara et.al. [4] in reference to agent capabilities. These requirements include expressiveness, abstraction, support for inferences, ease of use, application on the web, and avoiding reliance on keyword extraction and comparison. We believe these high level criteria are also relevant in the context of semantic web services but they do not address the specific requirements of dynamic web service discovery. To address these requirements, a capability language should provide:

1. The ability to declare what action a service performs.
2. The ability to allow a capability to have different sets of inputs.
3. The ability to declare preconditions and effects in some named rule definition language.
4. The ability to describe objects that are not input but are used or affected by the capability.
5. The ability to refer to ontological descriptions of the terms used in the description and thus place the use of the terms in context.
6. The ability to make explicit the domain or context in which the service operates.
7. The ability to classify capabilities based on aspects of the description enabling exact or partial matches between required and provided capability descriptions.

The genesis of the requirements is illustrated below, and we refer to them using this notation (req. 1) with the number corresponding to the requirement listed above.

There are several areas where we draw from existing work in capability description. We were influenced by the conclusions drawn in [5] and [6], where it was concluded after reviewing various description mechanisms and languages, that frame based representations were the most expressive and flexible means to represent the capabilities of intelligent agents. We start with an overview of case frames and look at several capability representations based on them, in the context of software agents and software reuse. Then we look at one of the current mechanisms for describing web services to see how well it provides a description of service capabilities.

Case Frames. [7] Much of the work in agent capability description has been based on the work of Charles Fillmore. We briefly review this work to understand why case frames are used to describe what agents and services do, and how they have been adapted over time.

Fillmore proposed a structure, called a case frame, to describe information about a verb. Each case describes some aspect of the verb and a completed case frame describes an action, its context and its consequences. The case frame provides a mechanism to state the who, what, where, when, how questions for actions. Fillmore elaborated several cases and postulated that other cases may exist. The base cases he described for verbs or actions are:

- Agentive - who does the action.
- Dative - who it happens to.
- Instrumental - what thing is involved.
- Factive - the object or being resulting from the action.
- Locative - where the action happens.
- Objective - things affected by the action.

In the context of semantic web service descriptions the case frame provides a convenient way of structuring the description of what behaviours, actions or capabilities a service provides.

For the purpose of representing service capabilities within a case frame structure, we can assume the agentive case implicitly represents the service itself and the dative case implicitly represents the service user, so these do not have to be elaborated explicitly. This does however imply that the capability description of a service is always from the perspective of what the service (as the agentive case) does or provides. For example, a service that provides goods that customers can buy, is a selling service. A service that finds and buys the lowest priced goods is a purchasing service.

The instrumental case represents things involved in the action. In the following paragraphs we will see how this has been used to represent the inputs for an action.

The objective case represents those objects that are involved in the performance of a service but not explicitly provided as inputs by the user. For example, a third party web service may offer to search for books by looking in the Amazon book catalog[3]. A user then may decide to use the third party service because they are unable to access Amazon directly and have heard that Amazon provides a competitively priced delivery service.

The factive case represents the results of the action and, in subsequent work discussed below, has been translated to represent the outputs or effects of an action.

In recent times Fillmore has been involved with the Berkeley FrameNet project[4], which is in the process of describing the frames (conceptual structures) of many verbs in common use [8]. The FrameNet system will be useful for the automated generation of descriptions, by providing base frames for many different kinds of service capabilities. For example the FrameNet frame for the verb sell contains the cases; *Buyer, Seller, Money, Goods, Rate, Unit, Means and Manner*. These are the possible cases for sell, therefore a description of a selling capability will need to incorporate some if not all of these cases to be effective.

2.1 Capability Descriptions for Software Agents

EXPECT [9,10] provides a structured representation of goals, based on verb clauses, it allows the representation of both general and specialized goals for agent planning systems. The structured representation allows reasoning about how goals relate to one another and allows inexact matching for loose coupling between representations of goals and capabilities descriptions. The representation is tied to a domain ontology to ensure the terms have consistent semantics amongst all users.

The verb clause consists of a verb and one or more roles or slots (cases). The role can represent objects involved in the task, parameters, or a description of the task. Roles can be populated by different types of objects including; concepts or instances from some ontology, literals, data types, sets, or descriptions.

[3] `http://www.amazon.com`

[4] `http://www.icsi.berkeley.edu/~framenet/`

An example (from [10]) of an Expect capability description for calculating the weight of objects is shown below:

```
((name calculate-total-cargo-weight-objects)
 (capability (calculate (obj (?w is (spec-of weight)))
   (of (?fms is (set-of (inst-of object))))))
 (result-type (inst-of weight))
 (method (sum (obj (r-weight ?fms)))))
```

When we apply the criteria from [4] noted above, this description would succeed on the expressiveness and inferences criteria but fail on ease of use and web applicability. Apart from keywords in the name "calculate-total-cargo-weight-objects", there is little in this capability description that could be used for discovery .

A consistent semantics is necessary for the representation of the structure of the capability description. However, in an open environment there will be many diverse contexts in which the capability description is used. This means a single ontology for the representation of the *content* of the description is not feasible.

The advantage of this description is the ability to use a rule notation to express conditions and effects. The use of rules to describe aspects of capabilities was also advocated in [6] where the ability to explicitly declare which rule language was being used is also provided (req. 3). The disadvantage of this capability description is that it would require training in order to write the descriptions.

Language for Advertisement and Request for Knowledge Sharing (LARKS) [11, 4] is a refined-frame based language for the specification of agent capabilities. It comprises a text description of the capability, with a structured description of the context, types, input, output, and constraints of an agent. An ontological description of terms used in the description can also be provided. The primary purpose of a LARKS specification is to allow inferencing and efficient accurate matchmaking.

In the current environment, where ontologies are proliferating, it is more likely that terms will be described by reference to external ontologies, rather than incorporated as an ontological description within the capability description itself (req. 5).

LARKS does not provide sufficient information in the form of a structured description of its purpose to enable discovery. The example below (from [4]) shows a capability description for a portfolio agent.

```
Context:            Stock, StockMarket;
Types:              StockSymbols = {IBM, Apple, HP},
                    Money = Real;
Input:              symbol:StockSymbols;
                    yourMoney:Money;
                    shares:Money;
Output:             yourStock:StockSymbols;
                    yourShares:Money;
                    yourChange:Money;
```

```
InConstraints:      yourMoney >= shares*currentPrice(symb);
OutConstraints:     yourChange =
                        yourMoney-shares*currentPrice(symb);
                    yourShares = shares;
                    yourStock = symbol;
ConcDescriptions:
TextDescription:    buying stocks from IBM,Apple,HP
                    at the stock market.
```

The information available for discovery is the unstructured text description "buying stocks from IBM, Apple, HP at the stock market", thus leaving keyword extraction as the only way of deciding what the service actually does. The capability of the agent being described is unclear, is it buying stocks on the stock market or directly from the company at the stock market?

In terms of the criteria, the language is expressive, and it allows inferences. It appears to be easy to use, although the example has an error[5] and other inconsistencies. This representation does not do well on the web applicability criteria but it has been used as the basis of the web accessible DAML-S Profile, which we look at in section 2.3.

The lack of an explicit action description means the capability has to be derived from keywords and unstructured text descriptions, but the advantages of this description mechanism are the ability to refer to ontological description of terms, comprehensive coverage of constraints and effects (rules) (req. 3), input/output (data), and the context the service operates in (req. 6). The ability to "matchmake" (req. 7) based on IOPE's was reported in [11].

2.2 Capability Description for Reuse

Web services are software, so we draw on work that has been done in the context of describing the capabilities of reusable software.

Reuse of Software Artifacts (ROSA). [12] The ROSA system is used for the automated indexing and retrieval of software components. In contrast to faceted classification, ROSA uses a conventional case frame structure, along with constraints and heuristics, to automatically extract lexical, syntactic and semantic information from free text software descriptions.

The automated interpretation of descriptive phrases into case frames makes this a potential tool for the generation and indexing of web service capability descriptions. However, apart from a few papers preceding [13] we have not been able to access this promising resource. Similar work has been reported in [14], where a web interface allows the entry of natural language descriptions of required components.

In terms of the criteria ROSA is expressive and capable of supporting inferencing. Comparisons are easily made between the "normalized" internal representations. ROSA is easy to use as the descriptions can be made in free text

[5] Input *shares* has to be a quantity to have meaning in the OutConstraint *yourChange*

and automatically translated into a structured description. The use of WordNet implies the use of other ontologies could also be supported (req. 5).

ROSA, being intended for the manual discovery of reusable software assets, rather than global automated web service invocation, does not deal with the possibility that some capabilities may be context dependent, if that is the case then the context should be made explicit (req. 6).

2.3 Web Service Description

DAML-S Profile [6] [15,16,17] builds on work on LARKS and ATLAS[7]. The DAML-S profile is a "yellow pages" description of a service, it is intended to specify what the service requires and what it provides. The service capability is described in terms of input and output parameters, preconditions and effects (IOPEs). The description also includes the service profile name, a text description, a reference to a Process specification (how it works), a service category (NAICS etc), and a quality rating. The profile allows the definition of service parameters to describe (non-functional) aspects of the service such as "MaxResponseTime", and information about the service provider or requester, such as their name, title, address, web URL etc.

The DAML-S Profile has further refined the basic case frame down to the description of capabilities only in terms of IOPEs. In the process it has lost the ability to explicitly declare what the service actually does. It has also lost the ability to describe the objects that are used but are not inputs in the description of the service [18].

To illustrate several points the extract below has been taken from the DAML-S V0.7 Congo Book example service profile description. The example represents the information in the profile that is machine processable and the types of the IOPE's.

```
serviceName              Congo_BookBuying_Agent
textDescription          This agentified service provides
                         the opportunity to browse a book
                         selling site and buy books there
NAME                     TYPE
(Inputs)
bookTitle                xsd:string
signInInfo               CongoProcess:SignInData
createAcctInfo           CongoProcess:CreateAcct
creditCardNumber         xsd:decimal
creditCardType           CongoProcess:CreditCardType
creditCardExpirationDate    time:TemporalEntity
deliveryAddress          xsd:string
packagingSelection       CongoProcess:PackagingType
DeliveryType             CongoProcess:DeliveryType
```

[6] http://www.daml.org/services/daml-s/0.7/

[7] http://www-2.cs.cmu.edu/~softagents/larks.html, http://www.daml.ri.cmu.edu/index.html

```
(Outputs)
EReceipt                CongoProcess:EReceipt
ShippingOrder           CongoProcess:ShippingOrder
AccountType             CongoProcess:CreateAcctOutputType
(Preconditions)
AcctExists              CongoProcess:AcctExists
CreditExists            CongoProcess:CreditExists
(Effects)
BuyEffectType           CongoProcess:BuyEffectType
```

The lack of an explicit means of declaring what the service actually does means that keyword extraction from the service name and description is necessary to discover the service's capabilities (req. 1). In this example, this is made more difficult by the description which states it is a BookBuying_Agent when it is a service that sells books.

In the text accompanying the example it is implied that the service provides two capabilities, "catalog browsing" and "selling books", neither of these are clear from the information provided in the form of IOPEs. In fact, there are also the implied capabilities to "accept credit card payments from the customer", to "check the customers credit availability" and to "deliver books to the customer". On one hand, it could be argued that these are capabilities that do not need to be exposed for discovery, on the other hand they are implicitly exposed by being declared as input parameters in the service profile.

The problem seems to be that the service profile does not have a mechanism to explicitly declare that a service may comprise several capabilities; and that each capability may have a different set of IOPEs. In addition, it needs to be able to hide the IOPEs that are not directly related to discovery.

Recent work by Sabou [19] in using DAML-S to describe a web service has revealed problems with describing services that may have alternative (sets of) inputs. The problem arises mainly in terms of the binding to WSDL, but it highlights the case where each capability provided by a service may have its own sets of alternative inputs (req. 2).

It is possible that the explicit declaration of capability has been replaced by the explicit declaration of the effects of the service. This is a valid modeling choice if it is used correctly. For example, instead of saying "we have the capability to sell books", the service could say "the effect of this service is that a book is purchased". The example does not support this interpretation.

In terms of the criteria, DAML-S has the potential to be sufficiently expressive to model both atomic and complex services and allows description at an abstract level. Being based on DAML+OIL/OWL it is implicitly capable of supporting inferencing and machine processing. The language appears easy to use well (and poorly). It is the best example of a web enabled language for the automated exchange and processing of capability information that we have looked at.

The shortcomings of the DAML-S Profile are the inability to describe the actual action performed (req. 1), and to describe the objects it may use or affect that are not provided as inputs (req. 4). It should also be possible to associate (sets of) inputs with specific capabilities (req. 2).

3 A Conceptual Model of Capability

In this section we present a conceptual model (figure 1) for capability descriptions. We believe that the model fully delivers all the requirements listed in section 2 and also satisfies the criteria proposed in [4]. In addition to the requirements and criteria, this model of capability descriptions is sufficiently detailed to facilitate the location of functionally equivalent or similar services. It is expressive enough to model simple atomic services as well as the functionality of complex, possibly composed, services. The model will allow the creation of capability descriptions that work human to human as well as machine to machine [20], because ultimately it is people who design and create software applications.

The capability description is presented as an Object Role Modeling (ORM) [21] model. ORM is a well known visual conceptual data modeling technique. The advantages of using ORM are that it has an associated modelling methodology and conceptual query language. It is implementation independent and has a formal semantics. It also has the advantage of being able to include a sample population, shown below the fact types, which helps to validate the model and demonstrate how it is used.

We briefly describe some of the main concepts of ORM is to assist the reader to interpret the schema presented below. The ellipses represent entity types (e.g. *Capability*), while the boxes represent the roles played by the entities in a fact type and a fact type can consist of one or more roles. Double arrows represent uniqueness constraints over roles (e.g. a *Capability* has at most one output Signature set), while solid dots represent mandatory role constraints (e.g. every *Capability* has an action *Verb*). A string, in parenthesis below the name of an entity type (e.g. *(id)*), indicates the presence of a value type with a name which is the concatenation of that entity type name and that string. In this case instances of the value type uniquely identify instances of that entity type (e.g. *Signatureid* is a value type providing identification for entity type *Signature*).

The sample population demonstrates the description of three capabilities. The first capability is the ability to book tickets for a performance of the opera "Carmen", at the Queensland Performing Arts Center (QTAC). The second capability provides valuations for pre-war Belgian stamps, and the third capability retrieves ontologies that contain a given string.

A *Capability* is described, in the first instance, by an action *Verb* that expresses what the capability does (req. 1). To allow for the fact that different verbs may be used to express the same action, synonyms are provided directly, and a definition is available in an *Ontological source* (e.g. dictionary, thesaurus, ontology, specification or standard) (req. 5). The ability to provide alternatives to the primary verb assists similarity matching of capabilities (req. 7).

From the case frame point of view, we have modelled cases as roles. We have grouped the roles according to the type of objects that play those roles. We distinguish between cases that play an informational role, such as *location, topic, manner* etc., from roles that are played by a *Signature*, and roles that are played by *Rules*.

A *Signature* represents a set of *Parameters*. A capability can have zero or more *input*, *uses* and *affects* signature sets, including the empty set (req. 2, 4).

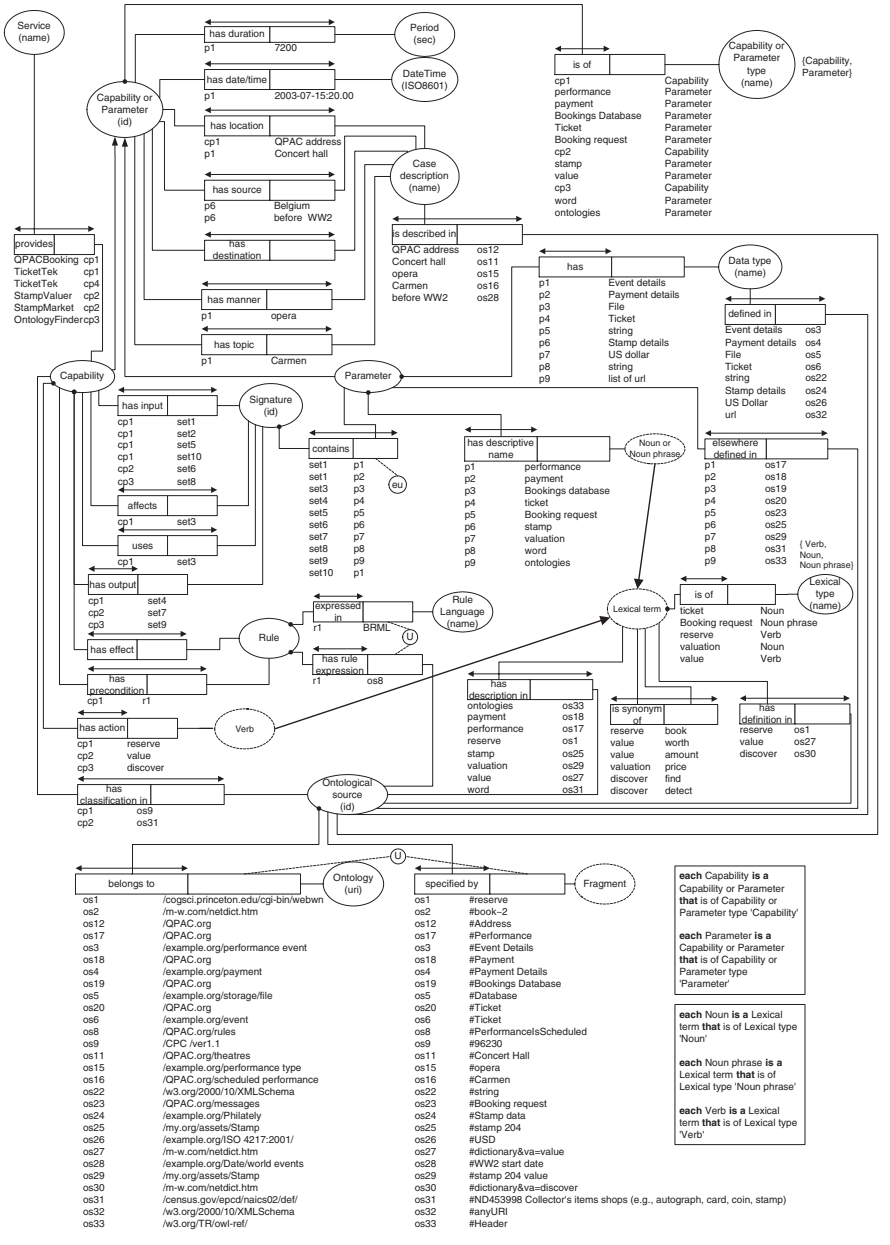

Fig. 1. A conceptual meta model of capability

For example, a service may take as input a name (string) and an age (integer), or an email address (URI) and an age (integer), or nothing at all. Each signature set must contain a different combination of elements, this is shown by the "eu"

constraint [22] on the role that connects to Parameter. Each Parameter and its associated *Data type* are defined in an Ontological source (req. 5).

The *output* role is constrained to have only one signature set, as we take the view that different output set would represent a different capability.

We created a supertype *Capability or Parameter* so we could share the definition of the informational roles, location etc. between the two types Capability and Parameter. These roles are played by a *Case description* described in an Ontological Source (req. 5).

We have distinguished the cases for preconditions and effects (PEs) and modelled these as roles played by Rules; as opposed to the input and output (IO) roles played by Signatures. This is because rules and signatures are fundamentally different and require a different treatment in the conceptual model. Each rule is associated with a named *Rule Language* and a rule expression (req. 3) in an Ontological source.

The use of an explicit domain or context identifier (req. 6) is provided by the role *has classification*. The classification itself is contained in an Ontological source.

An issue that may cause confusion, is that in this model we show verbs, nouns and noun phrases as subtypes of *Lexical term*. There is potentially a problem if the same word is used as a noun and as a verb. For example, the verb 'reserve' has 'book' as a synonym, however the noun 'book' would have a completely unrelated set of synonyms. We think this problem may be resolved by using a namespace identifier in conjunction with the word, rather than the abbreviated version shown in the model.

In terms of Fillmore's cases (section 2) the Agentive case is the service providing the capability, and the Dative case is the user, these are implicit. The Instrumental case is modeled as the *has input* and *uses* roles. The Factive case is modeled as the *has output* and *has effect* roles. The Objective case is shown as the role *affects*, and the Locative case can made explicit using *has location*, *has source* and *has destination* for both capabilities and parameters.

3.1 Evaluation of the Model

In this section we show how our capability description model satisfies the requirements listed in section 2.

1. The action declaration is explicitly provided by the role has action. The verb representing the action is defined in an ontological source. Alternative action words such as synonyms that are equivalent to the primary verb in this context can also be defined using the role has synonym. The explicit provision of alternative terms assists in service matching.
2. The model provides for different sets of inputs by allowing a capability to have different signatures for the roles has input, uses and effects. The signature is a possibly empty set of parameters. Each parameter is declared with a name (by convention the name should indicate its purpose) and its data type. However, the model does not only rely on descriptive parameter names it also allows both the parameter and its data type to refer to external definitions for more information.

3. Preconditions and effects can be defined by reference to an expression, using a named rule language, in an ontological source. The use of an explicit name for the rule language caters for the fact that web enabled rule languages are still being developed and until a clear favourite emerges, it is safer to explicitly state which one applies.
4. Objects that may be used or affected by the capability, but are not part of the input provided by the user, can be explicitly described using the uses and affects roles.
5. Most of the elements in the model can make reference to an ontological source for further information and clarification on how terms are intended to be used in the context of the capability.
6. The domain or context the capability operates within is made explicit by the has classification in role. Categorization schemes such as UNSPSC[8] and NAICS[9] can be used to describe the context in which a capability is performed.
7. The capability description provides many aspects that can be used for classifying capabilities. The has classification in role is similar to the level of classification available in UDDI. However, this capability description allows classification along much broader lines including the type of action performed, the location of the service, its manner of operation, and its topic of concern amongst others.

3.2 Querying the Model

A collection of capability descriptions can be easily queried using a conceptual query langauge like ConQuer [23,24]. An implementation of ConQuer is available in the Active Query tool, but we have been unable to access it yet, so the syntax shown below is based on the references above, rather than the output of the software tool. For users unfamiliar with ConQuer, the tick symbol is similar to the SQL select clause and these elements are returned or displayed. The +- symbol should be interpreted as "and", and alternatives are shown explicitly as "or".

The ability to query a collection of capability descriptions, based on the conceptual model, is of benefit to those users who have specific requirements beyond the types of input and output parameters provided by WSDL. Conceptual queries can access any of the objects and the relationships between objects shown in the model. The ability to make queries at the conceptual level will also be of benefit to service composers, allowing them to determine in advance what kinds of capabilities are available, and what kinds of objects a particular capability uses and has an effect upon in the performance of its function. This kind of information about the side-effects of a service or capability are important, and as far as we know are not available in any other structured service description mechanism. All of the major elements in the model provide the ability to access

[8] Universal Standard Products and Services Classification (UNSPSC), `http://eccma.org/unspsc/`

[9] North American Industry Classification System (NAICS), `http://www.ntis.gov/product/naics.htm`

further information in the form of ontologies, to assist with disambiguation and clarification.

Two examples of the types of queries the model can support are shown below.

1. Find a service that will allow me to book tickets for the opera "Carmen" and tell me when and where it will be held.

√ Service
+- provides Capability
+- has output Signature
+-contains Parameter has Datatype "Ticket"
+- has input Signature
+- contains Parameter
+- has descriptive name Noun or Noun phrase "performance"
or is synonym of Noun or Noun phrase "performance"
+- has manner Case description "opera"
+- has topic Case description "Carmen"
√ has date time DateTime
√ has location Case description

2. Find me a service that provides stamp valuations and show the type of input the service requires. Use the NAICs code 452998 that covers many types of specialist retailers including "Collector's items shops (e.g., autograph, card, coin, stamp)", to narrow the search.

√ Service
+- provides Capability
+- has classification in Ontological source
+- belongs to Ontology "www.census.gov/epcd/naics02/def/"
+- specified by Fragment "ND453998 Collector's items shops (e.g., autograph, card, coin, stamp)"
+- has output Signature
+- contains Parameter
+- has descriptive name Noun or Noun phrase "valuation"
or is synonym of Noun or Noun phrase "valuation"
+-has input Signature
contains Parameter
√ has Data type

4 Realisation

Various existing services and tools can be used to automate the generation of capability descriptions. The FrameNet frame [8] for the selected operation can be used as the basis of the capability description. Natural language descriptions [13,14] can be used along with WordNet verb synsets (groups of related terms)

to generate alternative verbs, nouns and noun phrases to populate the capability description.

Alternatively, the MIT Process Handbook[10] could be used to describe a capability as is, or with case refinements as described by Lee and Pentland in [25]. A capability could be declared to be equivalent to some process in the handbook, or it could be a specialization or generalization of a process description in the handbook. Klein and Bernstein [26] also suggest using the Process Handbook as a means to describe and locate semantic web services, and they provide the basis of a query language to use as an alternative to manual navigation of the handbook.

The more publicly accessible external classification schemes, standards, specifications, ontologies and other sources of information that are used in the capability description, the more likely it is that interaction partners will be able to find a common ground for understanding the terms the service uses.

5 Conclusion

The capability description we introduced in this paper can be used to advertise the capabilities of web services. The structure can also be used by service composers and planners to describe what they expect services to provide. Service composition planners can use the conceptual query language to interrogate a collection of capability descriptions. In addition, this capability description can be readily translated into a machine processable ontology. An explicit structured description of service capabilities allows the dynamic location of services based on their advertised capabilities rather than keyword searches and this will improve the efficiency and effectiveness of the discovery process.

One issue that still needs to be addressed is the specialization (by extension or restriction) of capability descriptions for specific contexts. The semantics of this are complex, as a capability description could potentially be both extended with additional cases, and existing cases could have their range of values restricted or removed.

The capability description we propose is not trivial, it will require much greater effort on the part of those describing services. We believe this level of complexity is unavoidable if we want to be able to achieve the goal of automated ad-hoc interaction between web services.

References

1. Haas, H., Orchard, D.: Web Services Architecture Usage Scenarios, W3C Working Draft 30 July 2002 (2002) Available from: `http://www.w3.org/TR/ws-arch-scenarios/`, (11 March 2003).
2. Heflin, J.: Web Ontology Language (OWL) Use Cases and Requirements, W3C Working Draft 31 March 2003 (2003) Available from: `http://www.w3.org/TR/webont-req/`, (15 April 2003).

[10] `http://ccs.mit.edu/ph/`

3. O'Sullivan, J., Edmond, D., ter Hofstede, A.: What's in a service? Towards accurate description of non-functional service properties. Distributed and Parallel Databases Journal - Special Issue on E-Services **12** (2002) 117–133
4. Sycara, K., Widoff, S., Klusch, M., Lu, J.: LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. Autonomous Agents and Multi-Agent Systems (2002) 173–203
5. Wickler, G., Tate, A.: Capability representations for brokering: A survey (1998) Submitted to Knowledge Engineering Review, December 1999. Available from: `http://www.aiai.ed.ac.uk/~oplan/cdl/cdl-ker.ps`, (4 October 2002).
6. Wickler, G.J.: Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation. PhD thesis, University of Edinburgh, Edinburgh, UK (1999)
7. Fillmore, C.: The Case for Case. Universals in Liguistic Theory. Holt, Rinehart and Winston, New York (1968)
8. Fillmore, C.J., Wooters, C., Baker, C.F.: Building a Large Lexical Databank Which Provides Deep Semantics. In: Proceedings of the Pacific Asian Conference on Language, Information and Computation, Hong Kong, Language Information Sciences Research Centre, City University of Hong Kong, PACLIC 15 (2001)
9. Swartout, W., Gil, Y., Valente, A.: Representing Capabilities of Problem-Solving Methods. In: In Proceedings of 1999 IJCAI Workshop on Ontologies and Problem-Solving Methods, Stockholm, Sweden, CEUR Publications (`http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/`) (1999)
10. Gil, Y., Blythe, J.: How Can a Structured Representation of Capabilities Help in Planning? (2000) In AAAI 2000 workshop on Representational Issues for Real-world Planning Systems.
11. Sycara, K.P., Klusch, M., Widoff, S., Lu, J.: Dynamic service matchmaking among agents in open information environments. SIGMOD Record **28** (1999) 47–53 `citeseer.nj.nec.com/article/sycara99dynamic.html`, (1 February 2002).
12. Girardi, M.R., Ibrahim, B.: Using English to Retrieve Software. The Journal of Systems and Software, Special Issue on Software Reusability **30** (1995) 249–270
13. Girardi, M.R.: Classification and Retrieval of Software through their Descriptions in Natural Language. PhD thesis, University of Geneva (1995) Ph.D. dissertation, No. 2782.
14. Sugumaran, V., Storey, V.C.: A Semantic-Based Approach to Component Retrieval. The DATA BASE for Advances in Information Systems **34** (2003) 8–24 Quarterly publication of the Special Interest Group on Management Information Systems of the Association for Computing Machinery (ACM-SIGMIS).
15. Denker, G., Hobbs, J., Martin, D., Narayana, S., Waldinger, W.: Accessing Information and Services on the DAML-Enabled Web. In: Second International Workshop on the Semantic Web - SemWeb'2001, Workshop at WWW10, Hongkong (2001)
16. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H.: DAML-S: Semantic Markup For Web Services. In: Proceedings of SWWS' 01 The First Semantic Web Working Symposium, Stanford University, CA, USA (2001) 411–430
17. Paolucci, M., Sycara, K., Kawamura, T.: Delivering Semantic Web Services. In: Proceedings of the twelfth international conference on World Wide Web, WWW2003, Budapest, Hungary, ACM, ACM Press (2003)

18. Wroe, C., Stevens, R., Goble, C., Roberts, A., Greenwood, M.: A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. International Journal of Cooperative Information Systems **12** (2003) 197–224
19. Sabou, M., Richards, D., van Splunter, S.: An experience report on using DAML-S. In: Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW '03), Budapest (2003)
20. Kovitz, B.: Ambiguity and What to Do about it. In: Proceedings IEEE Joint International Conference on Requirements Engineering, Essen, IEEE (2002) 213
21. Halpin, T.: Information Modeling and Relational Databases: from conceptual analysis to logical design. Morgan Kaufmann Publishers, San Diego, CA, USA (2001)
22. ter Hofstede, A.H.M., van der Weide, T.P.: Deriving Identity from Extensionality. International Journal of Software Engineering and Knowledge Engineering **8** (1998) 189–221
23. Bloesch, A.C., Halpin, T.A.: ConQuer: A Conceptual Query Language. In Thalheim, B., ed.: Proceedings of ER'96: 15th International Conference on Conceptual Modeling. Lecture Notes in Computer Science v. 1157, Cottbus, Germany, Springer Verlag (1996) 121–133
24. Bloesch, A.C., Halpin, T.A.: Conceptual Queries using ConQuer-II. In: Proceedings of ER'97: 16th International Conference on Conceptual Modeling. Lecture Notes in Computer Science v. 1331, Los Angeles, California, Springer Verlag (1997) 113–126
25. Lee, J., Pentland, B.T.: Grammatical Approach to Organizational Design (2000) Available from: `http://ccs.mit.edu/papers/pdf/wp215.pdf`, (24 April 2003).
26. Klein, M., Bernstein, A.: Searching for services on the semantic web using process ontologies. In: Proceedings of SWWS' 01 The First Semantic Web Working Symposium, Stanford University, California, USA (2001) 431–446 Available from: `http://www.daml.org/services/daml-s/2001/05/`, (20 September 2001).