

Chapter 4

Capabilities: describing what services do

This chapter¹ addresses the problem of advertising web services capabilities in such a way that their discovery can be automated. Although the other phases of service interaction, such as evaluation, selection, negotiation, execution and monitoring are important, discovering the services that can solve problems is the first step.

Service *capabilities* are the actions performed or the information delivered by a service. The purpose of a capability description is to allow clients to discover services that perform particular kinds of operations in particular contexts to satisfy their needs. This thesis addresses the problems associated with ad hoc software to software interaction and in this context, free text descriptions of capability are not appropriate. What is required is a “well defined” capability description that has a balance between machine accessibility and human understandability, because humans will always be in the service description loop.

The capability description represents a kind of “yellow pages” description of a service. This means the description should allow capabilities to be classified in some way to permit discovery using those classifications. This categorization can be in relation to external classification systems such as UNSPSC². Or, it can be in terms of other aspects of the description, such as *what* action is performed, on *which* things is it performed, *when* or *where* is it performed and so on.

Explicit capability descriptions for services are necessary because WSDL and OWL-S do not provide the language primitives to describe the functions a service performs (see section 1.2). Those languages rely on the use of descriptive names in operation signatures and documentation elements that are not machine accessible to describe what the service can do.

The chapter begins by identifying criteria drawn from, and used in, the evaluation of several existing mechanisms for describing software capabilities. In section 4.2 a conceptual model for web service capability descriptions is presented and in section

¹Based on work presented in [123]

²Universal Standard Products and Services Classification (UNSPSC), eccma.org/unspsc/

4.2.1 the model is subjected to the same evaluation. Section 4.2.2 demonstrates how collections of capability descriptions can be queried.

4.1 Existing work in capability description

A set of criteria for evaluating capability description languages were proposed by Sycara et.al. in [148] in reference to agent capabilities. These requirements are:

- expressiveness
- abstraction
- support for inferences
- ease of use
- avoiding reliance on keyword extraction and comparison

These high level criteria are also relevant in the context of semantic web services but they do not address the specific requirements of dynamic service discovery. To address these requirements, a capability language should provide:

1. The ability to declare the action a capability represents.
2. The ability to describe the object the action is performed on or with. This may or may not be the same as an object that is required as input.
3. The ability to make explicit the domain or context in which the service operates.
4. The ability to classify capabilities based on aspects of the description enabling exact or partial matches between required and provided capability descriptions.
5. The ability to allow a capability to have different sets of inputs.
6. The ability to refer to ontological descriptions of the terms used in the description and thus place the use of the terms in context.
7. The ability to declare preconditions and effects in some named rule definition language.

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

Material has been drawn from several areas of existing work in capability description. The conclusions drawn in [161] and [162] were a large influence, where after reviewing various description mechanisms and languages it was suggested that frame based representations were the most expressive and flexible means to represent the capabilities of intelligent agents.

This section starts with an overview of case frames and looks at several capability representations based on them, in the context of software agents and software reuse. Then the OWL-S profile is reviewed to see what it provides for the description of service capabilities.

Case Frames [61] Much of the work in agent capability description has been based on the work of Charles Fillmore. This work is briefly reviewed 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 used or affected by the action.

In the context of semantic 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, it is assumed 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 implies 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 the instrumental case is used to represent the input to a capability.

The objective case represents those objects that are involved in the performance of a capability but not necessarily provided as inputs by the user. For example, “Search Amazon catalogue” and “Reserve theatre tickets” specify objects (databases) that give important context to the capability, but they are not user inputs.

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 project³, which is in the process of describing the frames (conceptual structures) of many verbs in common use [60]. 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

³www.icsi.berkeley.edu/~framenet/

cases: *Buyer, Seller, Money, Goods, Rate, Unit, Means* and *Manner*. A service providing a selling capability will need to incorporate some if not all of these cases in its capability description in order to be discovered.

4.1.1 Capability descriptions for software agents

EXPECT [147, 72] provides a structured representation of goals, based on verb clauses, it allows the representation of both general and specialised goals for agent planning systems. The verb clause consists of a verb and one or more roles or slots (cases). A 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.

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 (req. 4). The representation is tied to a domain ontology to ensure the terms have consistent semantics amongst all users.

An example (from [72]) 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)))))
```

Applying the criteria from [148] described above, this description would succeed on the expressiveness and inferences criteria but fail on ease of use.

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 [162] where the ability to explicitly declare which rule language was being used is also provided (req. 7).

A disadvantage of this capability description is that it would require training to read and write the descriptions. Another disadvantage is the broad range of object types (concepts, literals, data types etc.) that can be used to populate a slot. Each slot entry would have to be supported by an explicit declaration of what type is being used in that context to avoid data mismatch problems.

Language for Advertisement and Request for Knowledge Sharing (LARKS)

[149, 148] 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. A local 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. 6).

The example below (from [148]) 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 description provides information about the context (Stock, StockMarket), the objects that are used in the performance of the capability (StockSymbols and Money), as well as the inputs, outputs, preconditions and effects of the capability.

In terms of the criteria, the language is expressive, it allows inferences and it appears to be easy to use. LARKS has been used as the basis of the web accessible OWL-S Profile, which is looked at in section 4.1.3.

The problem with this description is that it does not provide an explicit indication of what the service does or allows the client to do, for example “trade” or “invest”. The lack of an explicit action description means the capability has to be derived from the unstructured text description or inferred from the output and effects.

The advantages of this description mechanism are the ability to refer to ontological description of terms, comprehensive coverage of constraints and effects (rules) (req. 7), inputs and outputs, and the context the service operates in (req. 3). The ability to “matchmake” (req. 4) based on IOPE’s was reported in [149].

4.1.2 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) [74] The ROSA system is used for the automated indexing and retrieval of software components. 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 Wordnet lexical database is used to provide alternative words and term disambiguation.

The automated interpretation of descriptive phrases into case frames makes this a potential tool for the generation and indexing of service capability descriptions. However, apart from a few papers preceding [73] this promising resource is not accessible. Similar work has been reported in [146], 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 “normalised” internal representations of terms. ROSA is easy to use as the descriptions can be made in free text and automatically translated into a structured description. The use of WordNet implies the use of other ontologies could also be supported (req. 6).

ROSA, being intended for the manual discovery of reusable software assets, rather than global automated service invocation, does not deal with the possibility that some capabilities may be context dependent. In the web environment the context, if relevant, should be made explicit (req. 3).

4.1.3 Semantic service description

OWL-S Profile [47, 5, 130] builds on work on LARKS and ATLAS⁴. The OWL-S profile is promoted as a “yellow pages” description of a service, specifying what the service requires and what it provides. The service’s capability is described in terms of input and output parameters, preconditions and effects (IOPEs).

The description includes the service profile name, a text description, a classification in a service taxonomy such as NAICS⁵, and a quality rating. The profile allows the definition of service parameters to describe (non-functional) aspects of the service such as “MaxResponseTime”.

Information about the service provider or requester, such as their name, title, address, URL etc. can be provided. This information, along with the reference to a Process specification (how it works) in the Profile makes the profile implementation dependent. An OWL-S profile cannot be viewed as a declarative capability description that could, potentially, be implemented by different service providers.

⁴www-2.cs.cmu.edu/~softagents/larks.html, <http://www.daml.ri.cmu.edu/index.html>

⁵North American Industry Classification System (NAICS), www.ntis.gov/product/naics.htm

The OWL-S Profile mirrors the refined case frame LARKS. However, unlike LARKS there is no context identification. It may be that the classification is intended to replace the context identifier, but specifications such as UNSPSC and NAICS cover such a broad scope that they lack the granularity necessary for capability discovery. Also unlike LARKS, the OWL-S profile cannot describe the objects that are used by the service but are not explicitly provided by the client [163]. The OWL-S Profile is similar to LARKS in so far as it does not have the facility to explicitly declare what the service actually does.

To illustrate OWL-S the extract below has been taken from the V0.7 Congo Book example service profile description⁶. 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
(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 leads to ambiguity as the description states it is a BookBuying_Agent and the associated documentation shows that it is a service that sells books.

In the documents accompanying the example it is implied that the service provides two capabilities, "catalog browsing" and "selling books", neither of these are

⁶www.daml.org/services/daml-s/0.7/

clear from the information provided in the form of IOPEs. The inputs imply capabilities to “accept credit card payments from the customer”, to “check the customers credit availability” and to “deliver books to the customer”.

As profiles may contain more than one capability and each capability may have a different set of IOPEs, the service profile needs to have a mechanism to link IOPEs to specific capabilities.

Recent work by Sabou [136] in using OWL-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. 5).

An alternative to the explicit declaration of the action performed by a capability, is the explicit declaration of the effects of the service. This is an effective way of describing a service from alternative points of view. For example, instead of saying “we have the capability to sell books”, the service could say one effect of this service is that “a book is purchased”, and another effect is “a book is selected from our catalogue”. The example does not make this clear.

In terms of the criteria, OWL-S has the potential to be sufficiently expressive to model both atomic and complex services. Being based on OWL it is implicitly capable of supporting ontologies (req. 6), inferencing and machine processing.

The shortcomings of the OWL-S Profile are the lack of an action identifier (req. 1) and a description of the objects it operates on that are not provided as inputs which provide a context for the action (req. 2). It should also be possible to associate (alternative sets of) inputs with specific capabilities (req. 5).

4.2 A conceptual model of capability

This section presents a conceptual model for capability descriptions. The model is shown in figures 4.1, 4.2 and 4.3. The figures are split due to space constraints and should be considered together as one complete model. The split has been made to complement the conceptual groupings of roles and each role represents a case in a case frame. Figure 4.1 describes *what* the capability is and figure 4.2 gives more detail on *how* the capability is realised by inputs, outputs, preconditions and effects. Figure 4.3 shows the information held in the Information sources for this model. The information source model was introduced in the previous chapter.

In the introduction (section 1.7) a futuristic travelling scenario was presented to illustrate various capabilities that could be represented by services. In this section, some of these capabilities and others are used to populate the model. The capabilities include:

- BAFlightInfo and flightInfo are both capabilities that deliver flight details
- ConvertCurrency converts an amount from one currency to another
- PurchaseHelper assists with purchasing electrical products
- CatSearch is for finding products in catalogues

- Evaluator appraises products against requirements

These capabilities are *provided by* services named Eby, Ebat, AustOnlineMusic, BazzaOnline, ChangiAirport and Bairports. The constraints on the *provided by* role allow each service to deliver more than one capability and each capability may be delivered by more than one service. This means that a capability description is effectively an implementation independent interface to functionality.

A *Capability* is described in the first instance by the *performs action* role expressed as a *Verb or Verb phrase*. The verb is defined in an *Information source*. The information source is shown on figure 4.1 with an inner dashed line to indicate that it is an external type. The information source related to this model is shown in figure 4.3.

As different verbs can be used to express the same kind of action, synonyms and equivalent terms can be provided in the information source (req. 6). The ability to provide alternatives to the primary action identifier assists similarity matching of capabilities.

Just as in linguistics a verb usually has an object so to, a capability usually takes place in relation to some object or in some context. This relationship between the action and its context is represented by the *action is performed with* role.

To illustrate this connection between actions and objects, these examples (and those in the following paragraphs) are taken from the travelling scenario used to populate the models, and some web service descriptions from xmethods⁷.

Access	<i>UEFA's official match schedule</i>
Validate	<i>email address</i>
Generate	<i>fictional sample data</i>
Convert	<i>currency</i>
Evaluate	<i>product details</i>
Provide	<i>flight information</i>

The *action is performed with* role and the next three roles are in a relationship with a *Case description*. A case description is described in an information source. The case description allows us to use a proper name or phrase within the model (to enable discovery) and to associate this name with synonyms and equivalent terms in the information source.

Some capabilities, particularly those represented by verbs of “giving” or “making”, also have an indirect object that “receives” or “partakes” in the action or gives it additional contextual information. These objects are represented by a ternary relation between a capability, a *Preposition* and a case description identified by (... *in context of ... operates on ...*). The preposition helps to define the nature of the relationship. Some examples will make this clear.

⁷www.xmethods.com

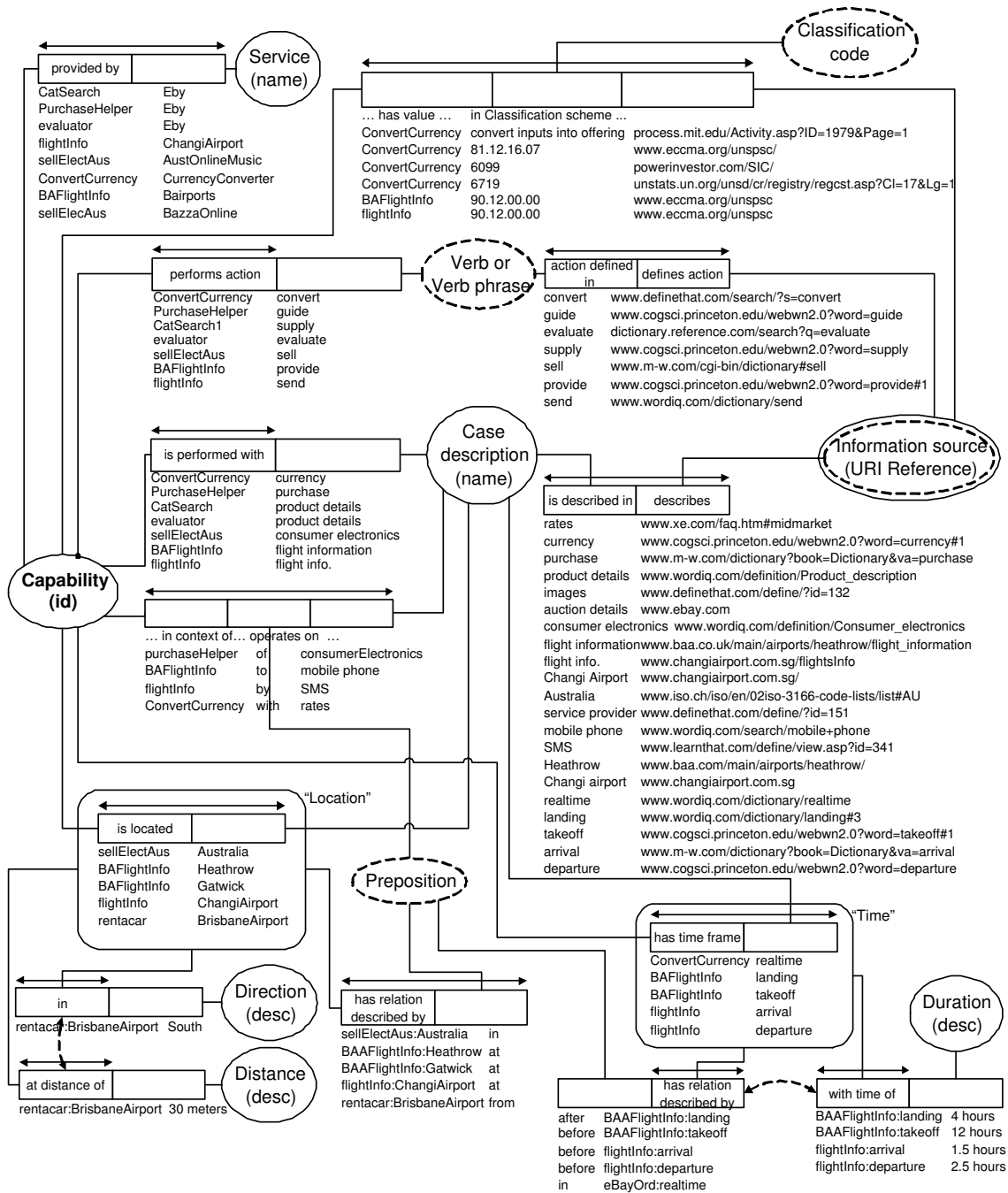


Figure 4.1: A conceptual model to describe capability

Returns	price of Book	at	BN.com
Send	SMS	to	mobile phone
Provides	data	from	national Medicare database
Guide	purchase	of	consumer electronics
Provide	flight information	to	mobile phone

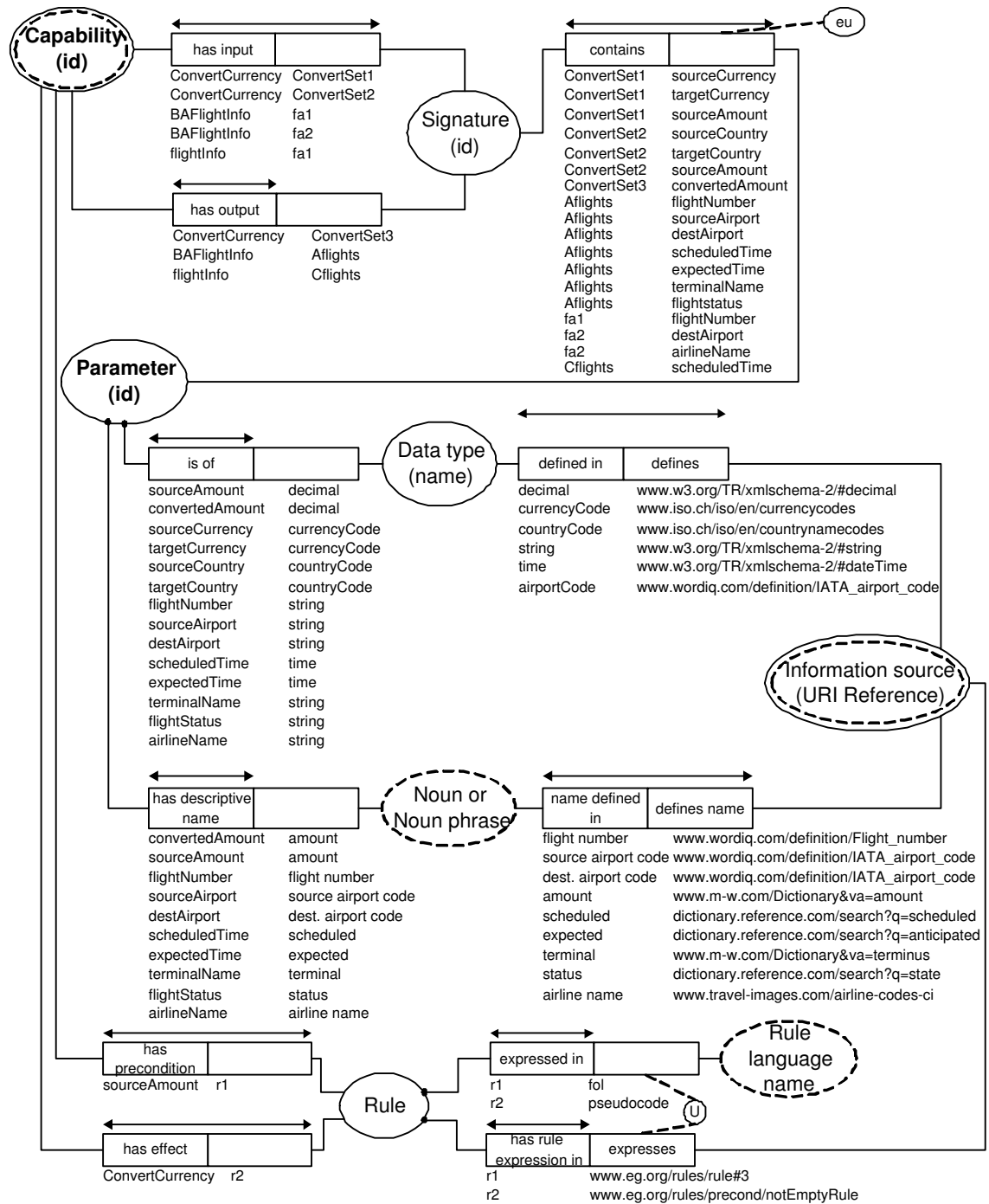


Figure 4.2: A conceptual model of capability parameters

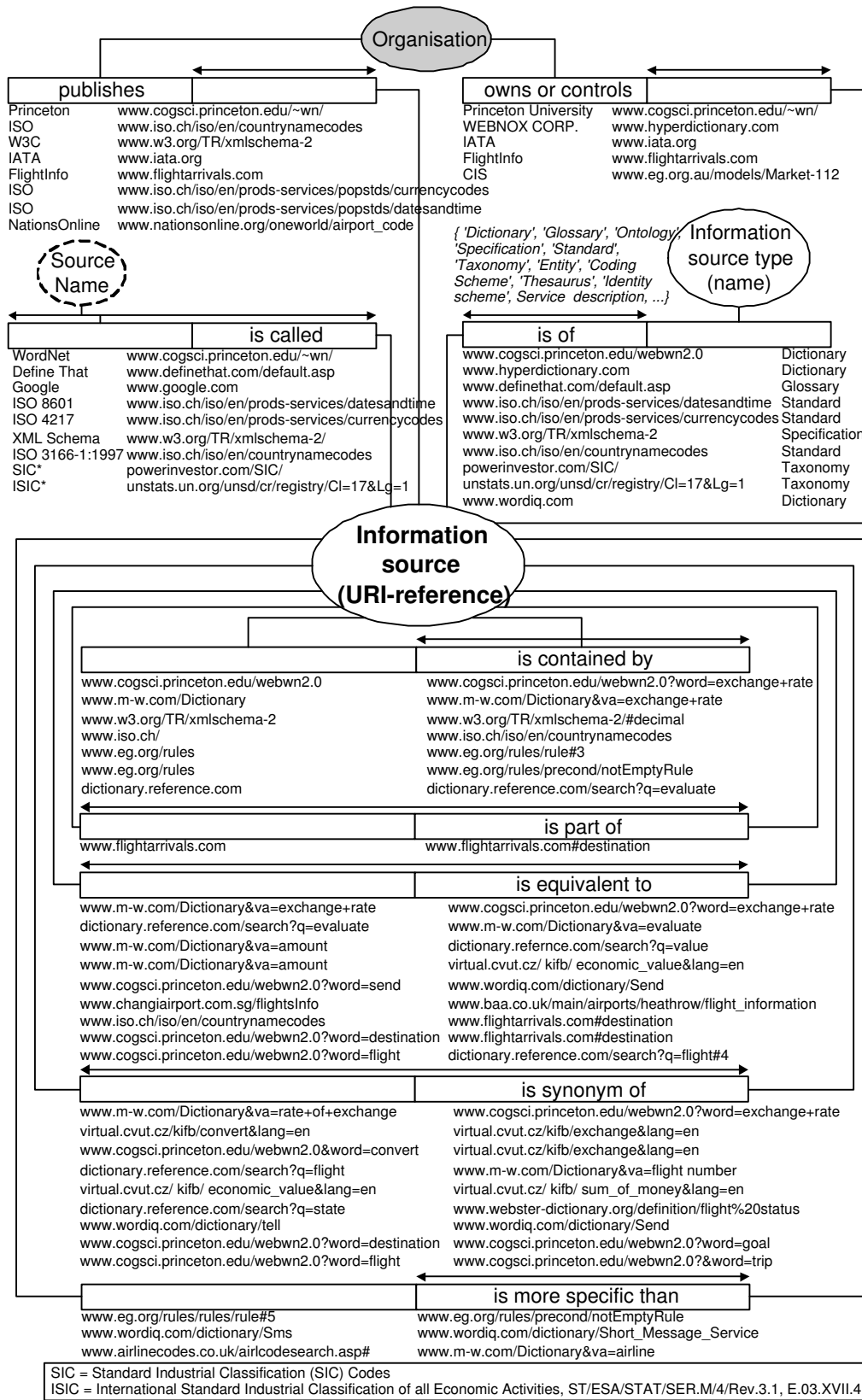


Figure 4.3: Information sources relating to the capability models

The “where” or locative case of a capability is represented by the objectified relation *Location*. By objectifying the relation, other information can be associated with a location to avoid ambiguity. The additional information consists of a preposition (in, at, to etc.), and directional information such as *Distance* and *Direction*. Some examples of locative information are:

Retrieve	tax rates	<i>in</i>	<i>USA</i>
Sell	consumer electronics	<i>in</i>	<i>Australia</i>
Search	Changi flight info.	<i>at</i>	<i>Changi airport</i>

Temporal aspects, or the “when” case is represented by the objectified relation *Time*. Once again, objectification allows us to associate additional optional time related information with the *has time frame* case description. Time concepts can be related to capabilities in various well known ways and some examples illustrate how this is done.

Get	prices	of metal and coin	<i>in</i>	<i>real-time</i>
Provide	flight information	<i>4 hours</i>	<i>after</i>	<i>landing</i>
Search	Changi flight info.	<i>1.5 hours</i>	<i>before</i>	<i>arrival</i>

Capabilities can be categorised in terms of external classification schemes with the ternary relation (... *has value ... in classification scheme ...*) between capability, *Classification Code* and information Source.

Chapter 3 discussed issues related to identifying instances in a global environment and pointed out that a classification code is only valid when it is linked with the scheme that issued it. The purpose of this ternary relationship is to make explicit such a link between a Capability its classification code and the scheme that issued the code. This relationship between code values and classification schemes applies equally to published classifications such as UNSPSC and NAICS as it does to less formal classification schemes such as the MIT Process Handbook or URI’s.

The roles shown in figure 4.1 describe the action and context of a capability in sufficient detail to allow discovery. To advance from discovery to evaluation and invocation, there are other aspects of capabilities that could be described. These aspects include, the inputs and preconditions that must be satisfied before the capability can be performed, and the output and/or effects of performing the capability. These elements are shown in figure 4.2.

The cases for input and output (IO) represented by a *Signature* are distinguished from preconditions and effects (PEs) represented by a *Rule*. This is because rules and signatures are fundamentally different and require a different treatment in the conceptual model.

A signature *contains* a set of *Parameters*. A capability can have zero or more alternative *input* signature sets, including the empty set (req. 5, 2). For example, a “person locator” capability may take as input: a name (string) and an age (integer), or an email address (URI) and an age (integer). Each signature set must contain a different combination of elements, this is shown by the “eu” constraint [152] on the role that connects to Parameter.

Each parameter has a *descriptive name* which is a *Noun or Noun Phrase* and *is of a named Data type*. Nouns and Datatypes are defined in an information source

(req. 6), which once again allows the association of equivalent terms and synonyms with the terms used in the capability description.

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

It is possible that input and output parameters may duplicate information already used to describe the context of the capability. For example, the capability “Returns price of Book at BN.com” has already described the nature of the output. The duplication is acceptable because it is important to have the action and its context to be fully described to enable discovery and also to associate specific inputs and outputs with descriptive parameter names and datatypes.

The preconditions and effects of a capability are represented by *Rules*. A precondition describes what must be true before the capability can be performed. Effects are what happens during and after the performance of the capability. There is no distinction between direct effects and side effects.

A rule is *expressed in* a named Rule Language and has a *rule expression in* an information source. Errors and exceptions can be also be represented with the *has effect* role.

In terms of Fillmore’s cases (section 4.1) the Agentive case is the service providing the capability. The Dative case is the user and is not shown on the model. The Instrumental case is modeled as the *action is performed with* and *has input* roles. The Factive case is modeled as the *has output* and *has effect* roles. The Objective case is shown as the role *is performed with*. The Locative case is made explicit using *is located* role and temporal aspects are described using the *has time frame* role.

The model delivers all the requirements listed in section 4.1 and also satisfies the criteria proposed in [148]. 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 complex or composed services.

4.2.1 Evaluation of the model

This section shows how the capability description model satisfies the requirements listed in section 4.1.

1. The action declaration is explicitly provided by the role *performs action*. The verb representing the action is defined in an information source. Alternative action words such as synonyms that are equivalent to the primary verb in this context can be defined in the information source using the role *has synonym*. The explicit provision of alternative terms assists in service matching.
2. Objects that may be used or affected by the capability, but are not necessarily input by the user, can be described using the *is performed with* and *...in context of ... operates on ...* roles.

3. The domain or context the capability operates within is made explicit by the *has classification in* role. Any categorization scheme can be used to give the context in which a capability is performed.
4. The capability description has many roles that can be used for classifying capabilities. The *has value in ... classification scheme ...* in role is similar to the level of classification available in OWL-S. However, this capability description allows classification along broader lines including the type of action performed, and the type of information used for the action.
5. The model provides for different sets of inputs by allowing a capability to have different signatures for the role *has input*. Each signature is a possibly empty set of parameters. Each parameter has a descriptive name and a data type. The model does not rely on these descriptive parameter names being understood outside of this context, so these names are further defined within an information source that allows the definition of synonyms and terms that are equivalent in this context.
6. Most of the elements in the model are defined in an information source. This allows the explicit declaration of relations such as synonymy, equivalence, more or less specific and whole to part.
7. Preconditions and effects can be defined by rules which use a named rule language and reference an expression in an information 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.

4.2.2 Querying the model

A collection of capability descriptions can be easily queried using a conceptual query language like ConQuer [18, 19]. For users unfamiliar with ConQuer, the tick symbol is similar to the SQL select clause and ticked items 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 or 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 are defined in an information source, to assist with disambiguation and clarification.

Two examples showing the types of queries the model can support are shown in figure 4.4.

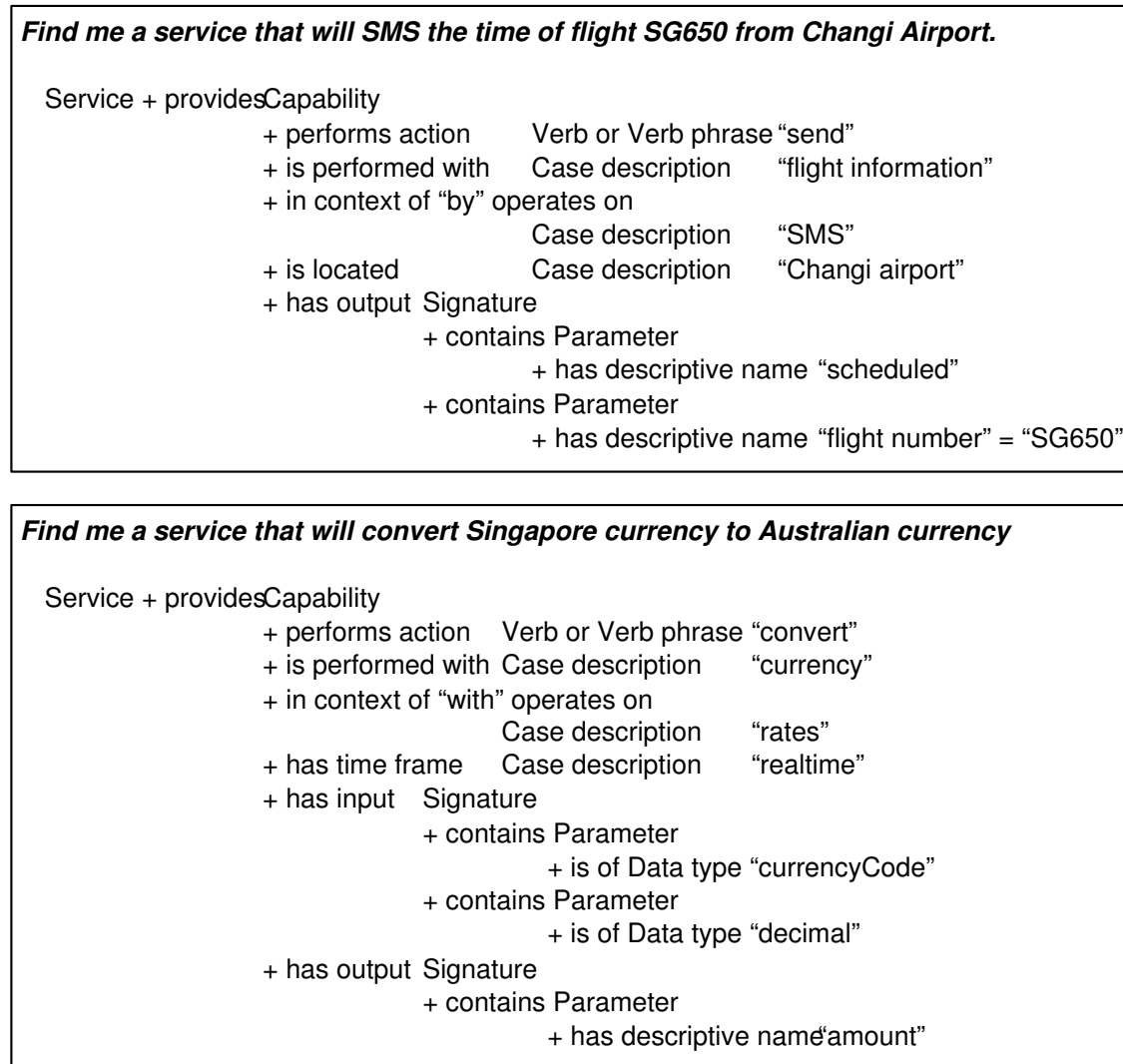


Figure 4.4: Conceptual queries on the capability model.

4.3 Realisation

Various existing services and tools can be used to automate the generation of capability descriptions. The FrameNet frame [60] for the selected action (e.g. sell) can be used as the basis of the capability description. Natural language descriptions [73, 146] 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.

The entries in the MIT Process Handbook⁸ could be extracted into a capability description perhaps with case refinements as described by Lee and Pentland in [103]. Alternatively, 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 [95] also suggest using the Process Handbook as a means to describe and locate semantic web services, and they provide 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.

4.4 Discussion

This chapter defined a structure for describing the functionality and context of services based on a case frame model. The capability model allows the description and runtime discovery of services based on what they can do, rather than the interface signatures they present to the world. The advantage of accessing services based on their advertised capabilities is that the client can engage services that provide capabilities that match their goals rather than interpreting or inferring capabilities from WSDL documents.

Capability descriptions can be presented at various levels of granularity, low granularity corresponding to WSDL operations, with higher granularity reflecting compositions of operations or services.

Service composition planners can use a conceptual query language to interrogate a collection of capability descriptions to find those that match specific criteria. The capability description format can be used by service composers and planners to describe their requirements for particular capabilities, leaving the implementation of these requirements to third parties.

An explicit structured description of service capabilities allows the dynamic discovery of services based on what they can do in a specific context rather than relying only on keyword based searches. This will improve the efficiency and effectiveness of automated service discovery.

The capability description can be readily translated into a machine processable ontology to take advantage of semantic search techniques.

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 made more general and more specific depending on which combination of cases is adapted.

⁸ccs.mit.edu/ph/

The capability description proposed is intuitive as it builds on common linguistic patterns. Although it will require more effort on the part of those describing services this is unavoidable if we want to be able to achieve the goal of automated ad hoc interaction between services.

An interesting area for further research not pursued here, is use of the capability description by (AI) planning agents or (data model) mediators working in the area of service evaluation, selection and composition. The mediator would be responsible for decomposing the client's requirements and selecting a combination of capabilities or services that provide the necessary data transformations and deliver the required functionality. The mediator would thus provide a more client-centric view which allows the client to request what it needs rather than the provider centric view proposed in this thesis.

Planning is related to the need to solve unique problems in a dynamic context. A typical example is "find all flights leaving Rome in the next two days that have a stopover in Bangkok". Services, on the other hand, advertise a clearly defined set of functions or capabilities in a clearly defined context. For example: (you can) find flights departing from Rome today, find flights departing from Rome on a specified date, find flights that stopover in Bangkok, and so on.

The purpose of explicitly advertising capabilities is to ensure clients know what a service will do in a given context and the limits of the flexibility the service is willing to offer. It is up to planners to find combinations of services and capabilities to solve their problems and integrate the information they can find into the form they need. It is not the responsibility of the service to change its advertised capabilities to suit the transient needs of the planner.

The (AI) planning concerns of how client goals are represented, how they are decomposed for discovery and matched to the capabilities delivered by service providers is out of the scope of this work.

Looking ahead: Capability descriptions are a foundation for the use of the interaction mechanism described in chapter 5. The capability description is the means by which clients can find, at runtime, the services that can provide the information or actions they require. Having found these services, the interaction mechanism allows the client to interact with any service without having pre-programmed calls to WSDL interfaces or having to interpret WSDL service descriptions and generate the appropriate calls on the fly.