

# Chapter 1

## Introduction

### 1.1 Problem area

Automated ad hoc interaction between web based applications is a desirable goal. Applications that can locate and interact with one another without human intervention will be able to achieve many tasks that are beyond human resources at present. These tasks can be as diverse as real-time monitoring and resource reallocation; repetitive polling for events that require a tactical response; and gathering of location dependent information for mobile devices.

The vision of real-time access to vast sources of information was described by Gelernter in 1991 [71]. Now, having achieved that vision with the internet, we are looking at how this evolving collection of information can be accessed and used. The number of providers and consumers and the sheer volume of accessible information means that the detection, collection, collation, processing and interpretation of information must be automated.

The next step requires a new kind of software with the ability to automatically process sources of information for many different purposes. We call this kind of software *services*. These sources of information will in turn be represented by *services* presenting information from resources as diverse as sensors, databases, catalogues, social and commercial activities including transport, health, shopping, banking, education and utilities. The possibilities of what services will be able to do is only limited by the ability of service developers to anticipate the needs of their potential clients.

The diverse nature of the information that can be made available by services for use by other services can be illustrated by an airport scenario. Imagine a not too distant future where a traveller will carry (internally or externally) devices that can find and relay information about the surrounding environment to them. These devices will be enabled by web services that can interact with other web services.

Every time the traveller arrives at a new airport their device or devices will find services that provide the information the traveller needs in this new context. For example, in Singapore the traveller may want to find an MP3 player, in Los Angeles she may need to eat, in Rome she may want to buy a handbag.

The information that the device finds in each place will be presented in a manner that is dependent on the location, purpose, culture, motivations and operating environment of the service providers. Some services will require customer specific details, such as tickets and baggage check numbers for baggage handling and checking-in.

Food outlets and shops will offer information about where they are located within airport buildings and the hours they are open as well as information about the products they offer for sale. The airport itself may offer services describing its utilities and facilities along with daily flight schedules. The airlines represented at the airport will also be able to provide current information about flights including the boarding time and gate number and status.

Some of the information available will have relevance only at certain times and some will be relevant in a specific place. For example, where to collect baggage for flight SQ236 in Changi airport terminal 2 vs. baggage collection for flight QF51 in terminal 2, or what food outlets are open at 5am in Rome's Leonardo Da Vinci airport. The same information, alone or aggregated with other information in different ways, may be provided by many different services.

In essence, the same type of information (e.g. flight details) may be available from different services. Each service provider may deliver different types of information (e.g. flight details and duty free shop opening hours). The information may be formulated in different ways (e.g. food outlets by type or food outlets by location). Services may be specific to certain types of places (e.g. airports vs. sporting venues) or particular places (e.g. Los Angeles international terminal 1 vs. international terminal 3). The information may be available all the time (e.g. coffee shop opening hours) or at different times (e.g. details for flight SQ235 to Brisbane on June 23 2004). The information may be freely available, restricted to registered users, or the user may have to pay to access information.

Here is a picture, not only of the potential diversity of what services can do or the information they can provide, but also of the diversity of providers, contexts and the ways of formulating information. In order for the devices held by the traveller to be able to handle this diversity, they will need to interact with many different kinds of services performing tasks in different ways. This gives us two roles for services, the first is the "traditional" view with the *service as provider* of information or functionality. The other view is of the *service as client* where a service elicits information and operations from other services; many services will play both roles.

At this point we need to explain why these services are not called *web services*. Web services are well known but they cannot provide the dynamic discovery and ad hoc interaction required in this scenario. The following section explains why this is so.

## 1.2 Web services

Web services are programmatic interfaces to functionality intended for application to application use across enterprise and web boundaries. Web services have gained

momentum in the last five years because the development of XML<sup>1</sup> has made it easier for systems in different environments to exchange information<sup>2</sup>.

There are two main strands of interest and activity in the web services area. The first strand is the focus of several industry organizations including the Web services Interoperability Organization<sup>3</sup> and World Wide Web Consortium (W3C) Web services Activity<sup>4</sup> which includes the Web services Description<sup>5</sup> working group. These web services are described using the Web Services Description Language (WSDL) [40]. They can be discovered by searching a UDDI<sup>6</sup> registry or by direct interaction with the service provider.

Semantic web services represent the second strand of web service activity. They are the focus of the OWL-S Coalition<sup>7</sup> [5, 31] and the Semantic Web Enabled Web Services (SWWS)<sup>8</sup> [32, 55] group.

### 1.2.1 WSDL

WSDL has become the de-facto standard for the description of Web services. It provides an interface definition language (IDL) in XML. A WSDL service defines one or more operations and the messages that can be sent and received by those operations according to a specified message pattern [77]. WSDL makes use of XML schema datatypes for the description of data. WSDL does not mandate any conventions for giving descriptive element or operation names to help describe the purpose of the service. WSDL does not provide facilities for describing the semantics of terms used in a Web service description [22].

WSDL web services are being used to automate interactions between business partners, similar to EDI<sup>9</sup>. These business partners make off-line agreements on syntax and semantics of the data being shared and the order of operations. This is necessary because data semantics and complex message ordering constraints cannot be represented in WSDL.

Web services can also be used outside of existing business relationships, such as those provided by Amazon<sup>10</sup> and Google<sup>11</sup>. To use services in this way, developers find the WSDL file and determine the data semantics and operation ordering from the names used in the interface and the documentation.

---

<sup>1</sup>[www.w3.org/XML/](http://www.w3.org/XML/)

<sup>2</sup>[www.w3.org/2002/ws/Activity](http://www.w3.org/2002/ws/Activity)

<sup>3</sup>[www.ws-i.org](http://www.ws-i.org)

<sup>4</sup>[www.w3.org/2002/ws](http://www.w3.org/2002/ws)

<sup>5</sup>[www.w3.org/2002/ws/desc/](http://www.w3.org/2002/ws/desc/)

<sup>6</sup>[www.uddi.org](http://www.uddi.org)

<sup>7</sup>[www.daml.org/services/owl-s/](http://www.daml.org/services/owl-s/), [www.daml.org/services/sws1/](http://www.daml.org/services/sws1/),

[www.w3.org/2002/ws/swsig/](http://www.w3.org/2002/ws/swsig/)

<sup>8</sup>[swws.semanticweb.org/swws](http://swws.semanticweb.org/swws), [www.deri.ie/index.html](http://www.deri.ie/index.html)

<sup>9</sup>[en.wikipedia.org/wiki/EDI](http://en.wikipedia.org/wiki/EDI)

<sup>10</sup>[www.amazon.com/gp/browse.html/104-8263098-0170340?node=3435361](http://www.amazon.com/gp/browse.html/104-8263098-0170340?node=3435361)

<sup>11</sup>[www.google.com/apis/](http://www.google.com/apis/)

Many of the large software providers such as Microsoft, IBM and BEA provide development platforms that insulate developers from much of the direct interaction with WSDL files, but it is still the operation and element names and documentation that are used to describe the what the service can do, what it needs, in what order, and what it provides.

WSDL gives service developers the means to provide “well defined”, “machine-accessible” “standard interfaces” [53]. Well defined interfaces are a direct consequence of using WSDL to structure the required information. Any syntactically correct WSDL service description will be well defined simply by using WSDL. Machine accessibility refers to the fact that XML is a machine readable language. Standard interfaces are more interesting.

The premise of *standard interfaces*, is that services providing the same information, or providing the same type of service can use the same interface (i.e. the same operations and operation signatures). This would allow web services as clients to be programmed in advance to access one standard interface regardless of who is providing the service. These interfaces would be defined by software providers and industry groups, based on a consensus agreement of what a service should do or provide in a particular domain, and the terminology that should be used to describe it.

Superficially this seems a reasonable way to deal with the problem of interacting with heterogeneous software services. The assurance of a standard interface guarantees that service providers and users are operating with the same set of information in a specific format. This paradigm has worked well for plug and play hardware devices and software libraries but it may limit the potential ubiquity and diversity of web services.

Standard interfaces may limit innovation by restricting the operations that services can expose. Service providers who can provide more operations than the standard interface allows, or do the same thing cheaper or quicker than other services cannot differentiate themselves from other services implementing the same interface. Alternatively, a service provider may do less than the operations defined in the standard, or they may provide some operations from one standard interface and some operations from another. Those providers, for whom the standard interface is inadequate or inappropriate have two choices. Either, only provide operations that conform to the standard, or provide non-standard interfaces; thereby reducing their set of potential interaction partners to those who can handle non-standard interfaces.

One problem that the “standard interfaces” solution does not deal with, is that many services will be so unique or specialised or generate so little revenue or are offered by so few providers, that it will never be worthwhile (for standards developers) to define standard interfaces for them. Or, it may be that different people, software developers in particular, like to do things differently. For these reasons, many if not most service providers will create unique one-off interfaces that suit their unique context. Diversity is the nature of the domain, it must be accepted and supported.

WSDL is a viable solution where two business entities can make explicit off-line agreements on the types and semantics of the data to be exchanged and the correct order of invocation of service operations can be encoded in the client software. However, the diversity outlined in the airport scenario for only one kind of location, could not be handled by a web-service enabled device using the current web service technologies WSDL (including message patterns) and BPEL4WS<sup>12</sup> (BPEL).

Using WSDL and BPEL, the developers' of our travellers devices can only ensure that a device can access web services at runtime by finding before deployment the standard (WSDL) interfaces of all the web services that operate in airports, as well as the interfaces of all web services that provide the other kinds of information the traveller might require. Having found these interfaces, the developer must handcraft calls (or sequences of calls) to those services. This is a non-trivial task, and it must be repeated for every domain the traveller may enter. For example in one city the traveller may attend a sporting event, in another they may attend a concert performance, in another they may go shopping. The means to access all the services and information that relate to those kinds of places will also have to be incorporated into the device. Devices will become overloaded with the means to interact with services they may never have to use.

### 1.2.2 Semantic web services

Semantic web services are on the boundary between the Semantic Web<sup>13</sup>, web services, and intelligent agents. Semantic markup is used in conjunction with WSDL and UDDI to provide a greater range of automated discovery, invocation and usability. Semantic web services are supposed to provide functionality to previously unknown interaction partners. The nature of what they provide, and how to interact with the service will be contained within machine readable semantic documents [114, 139, 142].

Semantic web services are described using OWL-S<sup>14</sup>. OWL-S is a three part semantic service description language based on the Web Ontology Language (OWL)[141]. OWL and consequently OWL-S have an XML syntax which gives machine accessibility to OWL-S service descriptions.

An OWL-S Profile describes what the service can do, an OWL-S Process model describes how the service achieves its purpose and the interaction protocol. An OWL-S grounding specifies how the inputs and outputs of a process are transformed into WSDL messages. OWL-S has been in development since May 2001 and has achieved a high level of acceptance in academia.

The ongoing work of developing the OWL-S semantic service description language is being carried out by the OWL-S Coalition with input from the The Semantic Web Services Initiative (SWSI)<sup>15</sup>. The OWL-S coalition maintains a mailing list

---

<sup>12</sup>[www-106.ibm.com/developerworks/webservices/library/ws-bpel/](http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/)

<sup>13</sup>[www.w3.org/2001/sw/](http://www.w3.org/2001/sw/)

<sup>14</sup>[www.daml.org/services/owl-s/](http://www.daml.org/services/owl-s/)

<sup>15</sup>[www.swsi.org/](http://www.swsi.org/)

for discussion, announcements and questions at [lists.w3.org/Archives/Public/public-sws-ig/](http://lists.w3.org/Archives/Public/public-sws-ig/).

The OWL-S service profile is the public interface of a service. The profile describes a service's capabilities in terms of a *transformation* of Inputs into Outputs and Effects, given certain Preconditions (IOPEs). Input and output data elements are described as instances of OWL classes described in some ontology. The nature of the transformation is not explicitly stated so it must be inferred from the output and/or effects, or the names used for elements in the description, or the textual documentation that is not machine accessible.

The problem with inferred capability can be illustrated by an example. Suppose a service is found where the input is the name of a book and a credit card number, the output is an invoice and the effect is that a book is delivered. We could infer from the transformation of inputs to outputs that this is a book selling service. But, it would be equally valid if we inferred that it was a library lending service, or an online document delivery service. Describing a transformation without describing the purpose, nature or context of that transformation introduces a high level of ambiguity.

A description in terms of outputs and/or effects describes a capability from a solution oriented point of view. This perspective allows different views of the functionality of the service. However, users and developers think also in terms of the problem and its context rather than the result that will be delivered by a solution. The implicit transformational approach to capability description may not serve those with a problem oriented perspective.

In recent developments observed on the mailing list and in papers such as [30] it appears that the OWL-S profile is being deprecated in favour of the process model for advertisement and discovery. This move away from the profile means that OWL-S services are advertised on the instance level according to how a capability is realised (process model), rather than a higher level declarative (reusable) view of what can be achieved.

There are several reasons why services described with WSDL or OWL-S cannot be used in the airport scenario. Firstly, because neither WSDL or OWL-S provide the ability to declare the what a service can do or its functional capabilities, these service descriptions cannot be used for dynamic discovery. Secondly, WSDL and OWL-S services only use local parameter names to describe the data they exchange with clients. This is not sufficient for ad hoc interaction where the participants have no prior agreements in place over the syntax and semantics of the data.

In the next chapter (2) WSDL and OWL-S are evaluated to gauge how well they perform in their roles as interface definition languages. The results of the evaluation give further weight to the argument against WSDL and OWL-S as languages capable of enabling ad hoc interaction.

## 1.3 Ubiquitous services

Ad hoc interaction is the ability of services to discover and invoke other downstream<sup>16</sup> services in order to provide composite and complex functionality dynamically. Ad hoc services, will need both reactive and proactive interaction abilities depending on their users' (human, agent, or other service) goals, and the resources they have available.

The promised mobility, ubiquity and diversity of services and service enabled devices in the future means that the developers of services and other software will not know in advance about all the services their application may need to use. These developers will not be able to rely on standard interfaces nor will they be able to determine at runtime, from WSDL or OWL-S interfaces, what capabilities are provided by another service. Another means of enabling ad hoc runtime interaction between services is required.

The main obstacle to advancing the vision of ad hoc runtime interaction is complexity. The complexity of services is related to:

1. The information the service requires and provides and the nuances of the domain or context the service operates on and in.
2. The specific nature of the operations the service provides and the constraints related to those operations.
3. The necessary ordering of operations to achieve the desired result.

The current web service description mechanisms place the responsibility for dealing with the complexity of a web service on the client. An alternative view is that the responsibility for handling complexity should be with the entity that understands it best, and this is the service provider, or its facade or mediator [70, 55] not the client.

To enable services to interact in a diverse ad hoc environment there must be a change from the current model of software as protected pieces of functionality that are accessed via well defined standard interfaces, to a model that allows services to proactively assist their users to achieve their goals. Services must take on a mediator or helper role to shield clients from their complexity. They must provide a way for clients to understand what function they perform. They must provide information about what data they need from clients in order to perform the function. Finally they must guide their clients through the necessary steps to achieve their goals successfully. This is the true meaning of "self describing web services" [154, 139].

## 1.4 Problem statement

The vision for electronic services in the future is for *loosely coupled* applications operating on different platforms *inter-operating without prior agreements in place*

---

<sup>16</sup>Upstream services are users or clients, downstream services are providers of functionality.

and *without human intervention* at runtime. There are three problems that must be addressed before this vision can become a reality. These problems are closely aligned with the three aspects of service complexity identified above.

1. Services and clients must share an understanding of *what* they are talking about.

Shared understanding of *shared data* is a problem inhibiting ad hoc interaction between services. The words, terminology and definitions used by service designers and developers to describe the information the service requires and provides, are naturally biased towards their own context and naming conventions. This local terminology may not be understood outside of the local environment.

2. Services and clients must share an understanding of *why* they are talking to one another.

Finding services that can assist clients achieve their goals is a problem inhibiting ad hoc interaction between services. The current web service description languages do not allow providers to describe what their services actually *do*. This means that service clients cannot locate providers to gain immediate access to the best possible solutions to their problems.

3. Services and clients must share an understanding of *how* they talk to one another.

The lack of a shared means of communication is a problem inhibiting ad hoc interaction between services. At present web services lack the means to *exchange* meaningful information with one another. Services are unable to *cooperate* in an ad hoc manner to solve problems.

### 1.4.1 The problem of shared data

The terminology used by designers and developers to name application elements is usually dictated by personal preferences and local coding conventions [51], or organizational and cultural requirements [68]. Ideally, terms are named to reflect the purpose or type of the element but even these conventions allow many different variations especially when combined with the technique of “mashing” words together. In the world of web services it is no longer possible to assume that all clients will share the local conventions of the service provider.

The selection (and conjunction) of terms is often context dependent, for example a travel application would use the term “arrival date” (or “arrivalDate” or “arrival\_date”) whereas a delivery tracking application would use variations of “delivery date” to represent the same concept.

The datatypes for elements representing the same concept can also vary. For example, dates can be represented with explicit date datatypes such as XML Schema date (20040520), or strings “May 20 2004” (or “20/5/2004” or “5/20/2004”). Or

they could be represented as combinations of integers representing the year, month and day.

These kinds of variations mean that two randomly selected web services will probably use different terms or conjunctions of terms in combination with different datatypes to represent the same kind of element. This data mismatch [56] is a serious problem for service interoperability.

It is suggested that one way to address the problems associated with sharing data is to create “global” or “universal” definitions. Nowadays these are constructed as ontologies using an ontology language such as OWL. Ontology is a word that has come from philosophy via AI knowledge bases into general use in the last few years. “An ontology is a shared and machine-executable conceptual model in a specific domain of interest” [32]. Ontologies are a way of recording and sharing a model or schema of the objects in a domain and the relationships between them [76]. Ontologies have much in common [93, 115] with more traditional conceptual modeling techniques such as Entity Relationship (ER) diagrams [39] and Object Role Modeling (ORM) diagrams [83].

There is a view of ontologies as repositories of universal or global definitions of objects and their relationships that all the applications in a particular domain can commit to and share [34, 147, 106]. This view is losing traction as is becoming clear that ontologies reflect the point of view and concerns of their creators [155, 68, 28]. Hence, they usually represent context dependent views rather than universal ones.

An alternative to global ontologies, is the likelihood that software vendors and other domain specific interest groups will generate ontologies and specifications to suit their own requirements and implementations. In this case, web service providers risk a problem similar to vendor lock-in; by selecting one particular ontology over another, service providers reduce their client base to those that adhere to the same model.

Patil et. al. [131] report that an experiment to annotate publicly available WSDL service descriptions with semantic meta-data [135] using domain ontologies, had to be curtailed because there were only two domain ontologies available that they could use. This reveals the problem that even if domain ontologies could represent all things to all users in the domain, these domain ontologies do not exist.

One way of linking ontologies to provide global or sharable views is to provide “mappings” [24, 49, 45, 109] from one ontology to another. Mappings provide links between ontologies that express well known lexical, semantic and logical relations such as synonymy, equivalence, hypernymy and hyponymy (more or less specific), holonymy (whole to part) and meronymy (part to whole) relationships. This “contextualization” [23] allows the creation of context specific ontologies that provide links to other ontologies when an understanding of the objects and relationships they describe must be shared across context boundaries.

Application profiles [11, 68, 87] are another way of creating shareable descriptions of objects and their relationships for web based applications. Application profiles

use publicly available metadata schemas, such as the Dublin Core element set<sup>17</sup> to describe the elements they contain. An application profile contains elements (and relationships) which are described using a mutually understood terminology. For example, Dublin Core contains the element “Publisher” which is defined as “an entity responsible for making the resource available”. All profiles and ontologies that reference “dc:publisher”<sup>18</sup> adhere to this definition of the term in their context.

The initial work on application profiles focussed on collecting web accessible meta-data sets such as Dublin Core. Subsequent efforts [10] have brought other meta-data sets to web accessibility by creating namespaces [26] for them. This is done by assigning Uniform Resource Identifiers (URIs) [15] to these metadata sets and fragment identifiers for the elements they contain. For example, the MARC standard record format for library catalogues will be referenced by:

“<http://www.loc.gov/marc.bibliographic/>”

The assignment of a URI to a metadata set does not necessarily mean that there will be a retrievable document or machine-accessible representation of metadata elements at that URI, but it does allow mutual understanding and agreement about the source of the information.

There are many other sources of information already in a retrievable or machine-accessible form on the web. These sources (such as standards, specifications, dictionaries and thesauri) can be used to provide further information about the elements and relationships contained within ontologies and other specifications. One example of online information sources are topic maps<sup>19</sup>. Wilde’s WWW Online Glossary<sup>20</sup> is a topic map that provides information about an extensive list of acronyms. An application or ontology that uses an acronym, could give an (equivalence) mapping to the definition in Wilde’s.

Grounding local descriptions in global specifications provides the means to negotiate shared meaning with heterogeneous clients. Services as providers can make available alternative terms, grounded by external definitions, to enable clients to match the terms the service uses with their own local and grounded definitions.

The purpose of creating mappings and using references to meta-data sets is to remove ambiguity with redundancy [96]. Instead of relying on global one-size-fits-all ontologies, local ontologies, schemas and profiles need to explicitly provide the mappings that are relevant in that context. Redundancy goes some way to ensuring mutual understanding across context boundaries.

The advantage of using several publicly accessible definitions instead of local terms is that common ground can be found between services and clients to share an understanding of the service’s data requirements. Although the semantic web promises seamless inferencing to solve the problems of data mismatch [16], at some level there must be a mapping between the terms that are used by a service and

---

<sup>17</sup>[dublincore.org/](http://dublincore.org/)

<sup>18</sup>dc is a typical abbreviation to reference the specification at [purl.org/dc/elements/1.1/](http://purl.org/dc/elements/1.1/)

<sup>19</sup>[www.topicmaps.org/xtm/1.0/](http://www.topicmaps.org/xtm/1.0/)

<sup>20</sup>[dret.net/glossary/](http://dret.net/glossary/)

alternative definitions that are also relevant *in this context*; that is the context the service operates in and on.

### 1.4.2 The problem of advertising and finding services that can satisfy goals

In recent times the Semantic Web<sup>21</sup> [16], and Web services<sup>22</sup> 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 and UDDI. 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" [80]. The Web Ontology Language (OWL) requirements describe "serendipitous interoperability" as the ability of devices to "discover each others' functionality and be able to take advantage of it" [88]. This idea can also be applied to web services.

At present web services are described using WSDL which allows the description of web service interfaces as a set of operation signatures. There are two alternative ways of determining what the service can do from the WSDL description. The first way is for developers to manually search for services and read the documentation to see what the service provides, 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 latter approach is particularly prone to problems, as most software developers use names and words idiosyncratically and the words may or may not bear any relevance to what the service actually does.

A better way forward for web services is to advertise an explicit description of what they can do or provide, that is their capabilities or functionality, rather than operation signatures, so clients can find, evaluate and invoke services based on matching their needs with what the service provides. Functional capability descriptions allow clients to discover services that provide the functionality they require to satisfy their requirements or goals. This dynamic selection of service providers based on what they can do means client developers will not need to know in advance the API's of all the services that can contribute functionality to their applications.

The functional service description should be a black box view of *what* capability can be performed not a glass box view of *how* it is done [13]. The capability description provides a way of comparing functionally equivalent services [5]. There may also be non-functional aspects of services, such as cost and quality of service that may also need to be taken into account in the evaluation, selection, and configuration

---

<sup>21</sup>[www.w3.org/2001/sw/](http://www.w3.org/2001/sw/)

<sup>22</sup>[www.w.org/2002/ws](http://www.w.org/2002/ws)

of services following their discovery [160]. A description of these aspects is outside of the scope of this work and the reader is referred to [127] for a comprehensive treatment.

Following discovery, the matching of service capabilities to the client's goals or requirements can be performed using various matching techniques. One such matching algorithm from the OWL-S coalition is described in [128] and [148]. With this technique the matching goes beyond finding equivalent keywords representing the inputs and outputs of a service to look at the subsumption relationships between the terms used in service advertisements and the terms used in the clients' service request. The degree of match is computed and services with a higher match are selected.

The OWL-S coalition describes capabilities in terms of the *transformation* of inputs into outputs and effects, given some preconditions [128, 129, 148]. The OWL-S profile does not provide the facility to describe the nature of the transformation performed so the actual capability must be inferred from the results, this is discussed further in chapter 2.

There are other more explicit ways of describing the capability of a service. One way is to categorise the service in relation to a *taxonomy* of services [48] such as the United Nations Standard Products and Services Code (UNSPSC) [42]. The UNSPSC is a large and publicly accessible repository but its primary function is to describe products rather than services. The definitions of services it does contain, such as "Data conversion service", are too coarsely grained for ad hoc discovery. As taxonomies of services become larger it will be necessary to provide more finely grained distinctions that relate capabilities to particular contexts. This will eventually make taxonomies unwieldy and harder to use [145].

Another way of describing capabilities is to provide a *functional* declaration. Charles Fillmore [61] proposed a functional way of describing verbs using "cases" contained in case frames. A case describes a different aspect of the context of a verb for example: agent (who), location (where) and instrument (how). The case frame structure has also been used to describe the capabilities of software agents [161]. Conceptual graphs [144] use similar cases to Fillmore with a visual representation. Conceptual graphs can be translated from the visual representation to a formal logical description of actions.

The use of a limited number of cases to describe actions, leads to the idea that there may be a limited number of things that can be done by software. The verbs in the WordNet lexical database<sup>23</sup> can be used to look for the kinds of actions that can be performed by software. WordNet contains 15 verb clusters [120] including "change", "communication", "competition", "cognition", "perception", "bodily functions and care" and "emotion or psych" amongst others. Clearly there are some clusters that can be represented by software such as change and communication, and some that cannot such as bodily functions and care.

---

<sup>23</sup>[www.cogsci.princeton.edu/~wn/](http://www.cogsci.princeton.edu/~wn/)

The MIT Process Handbook<sup>24</sup> [110] took this idea to a satisfactory conclusion and derived<sup>25</sup> its classification of processes also from WordNet. The MIT Process handbook hierarchy is based on 8 verb categories: *Creation, Modification, Preservation, Destruction, Combination, Separation, Decision* and *Management*. These categories provide an intuitive way of grouping all the actions that can occur in business processes and hence all the actions that can be provided by web services. The use of these categories in the description of services could provide high level means of determining whether services provide the necessary functionality. It could also prove useful for service users and composers to have a way of expressing which kinds of actions are appropriate in a particular application context.

The advantage of a structured declarative capability description is that clients can select and use services based on an explicit description of the service's functionality in context and how well it meets their goals. Abstracting the description of service capabilities away from the description of interfaces makes it possible to reuse capability descriptions so that the same capability can be delivered by many providers, and each provider can deliver many capabilities.

The ability to differentiate and specialise service offerings is one of the foundations of competitive advantage [27]. Standard interfaces that deliver a one-size-fits-all capability description end up being satisfactory for very few providers [38] and reduce the competitive incentive to provide services for commercial gain.

### 1.4.3 The problem of interaction between web services

Currently software is created by developers with an explicit knowledge of the modules and libraries (services) that contribute to the functionality of the application under construction. This knowledge is encoded in the modules' or services' Application Program Interfaces (APIs).

An API represents a contract [117] which states something similar to "if you give me this type of information, under these circumstances, I will give you back this kind of result". An API specifies the operations that can be invoked and what is required by each operation (its input signature and preconditions) to generate the output or effect. An API may also include, as a commentary, the constraints on the ordering of operations, or it may provide an abstraction in the form of a "controller" operation that manages the ordering constraints without exposing the sub-operations to the developer.

In the context of ad hoc interaction, it will not be possible for all client services to be developed with advance knowledge of the API's of all the service providers they may have to interact with. A new mechanism is required to tell potential clients what the service (as provider) needs to know in order to perform its advertised capability.

In real life we do this by means of a conversation or dialogue that is based on well understood interaction primitives such as asking questions, giving answers and

---

<sup>24</sup>[ccs.mit.edu/ph/](http://ccs.mit.edu/ph/)

<sup>25</sup>[mitpress.mit.edu/books/chapters/0262134292chap8.pdf](http://mitpress.mit.edu/books/chapters/0262134292chap8.pdf)

providing more information or disambiguation when necessary to ensure mutual understanding.

A shared language for the exchange of information would allow services as providers to proactively seek the data they require from clients such as input and configuration parameters. This language could also allow clients to seek further information from the provider to ensure mutual understanding of the requirements. However, a language is not sufficient on its own, there must also be a way of interpreting, managing and generating messages in the language [54], such that the result of the interaction satisfies the client's goals.

Conversations and dialogues often follow repeatable patterns and for some problems in clearly defined contexts an interaction protocol can be developed. An interaction protocol is a plan that defines which type of messages can be sent in which order.

An example of interaction protocols in action are auctions such as the English auction. In an English auction, the problem to be solved is to maximise the amount the seller receives from one of a group of potential buyers. There are several implicit rules that contribute to the success of the protocol. Firstly, buyers know in advance that the only information the auctioneer requires from them is a bid. Secondly, it is assumed the buyer understands the auction protocol and has inspected the "lot" or product to determine its fitness. This means the buyer may not interrupt the enactment of the protocol with questions about the product or the protocol. Thirdly, the buyer knows that the auctioneer will notify them if they have made the winning bid.

Much of the work that has been done in the area of conversations for agents [43, 62, 2, 66, 125] and lately web services [12, 105, 84, 85, 97, 69, 151, 165] is directed at specifying the order of messages in a conversation in a similar manner to interaction protocols. The argument for conversation protocols is that if both sides are aware of the correct sequence of messages they can participate correctly in the interaction.

Conversation protocols are created as either state charts [86] with messages representing the transitions between states; or as AUML interaction diagrams [125, 113]; or as Coloured Petri Nets (CPN) [43, 116, 165]. A problem with these approaches is how to represent or translate the conversation protocol so it can be shared, understood, agreed upon and enacted by both parties at runtime. Another problem with the protocol approach, is the computational cost associated with the effort required to agree on which protocol to use [132] and to ensure runtime compliance [108, 156] without human intervention.

The digressions necessary for help and error handling are often not specified in conversation protocols. When these alternative paths are specified, they can cause the specifications to become overly complex [132]. This complexity impacts adversely upon human understanding of the interaction specification. The situation then arises, that developers who cannot understand the interaction specification,

cannot create software that can deal properly with the complexity at runtime. Several specific proposals for web service interaction languages are discussed in section 5.5.

In contrast to auctions, ad hoc service interactions will take place in a context where the nature and order of the information the provider requires is not known by the client. The client may need to question the provider to elicit further information or the client may need to tell the provider that it cannot supply the necessary information.

This means ad hoc interaction between heterogeneous services cannot be defined in terms of a pre-set pattern of message exchanges. It is the nature and sequence of the data requirements of the service provider that determine the direction of the dialogue. The service must dynamically construct the “interface” presented to the client at runtime based on the client’s resources [150].

The means to exchange information via conversational dialogue would enable services and their clients to interact with one another at runtime without prior knowledge of WSDL service descriptions. Thus reducing what the client needs to know in advance. Flexible dialogue style interaction between services also allows the use of techniques to provide dynamic solutions to problems when appropriate. For example help messages can contain pointers to other services for dynamic content generation or data manipulation [1].

There are two advantages of a dialogue approach to dynamic service interaction. The first is that clients do not have to know the service’s API (in WSDL) in advance or interpret the WSDL description on-the-fly at runtime. The second advantage is that clients using a conversational mechanism can interact with *any* service at runtime rather than being tied to specific service implementations via hard-coding to published WSDL descriptions.

## 1.5 Scope

The focus of this thesis is firstly on *what* services talk about and how the data required or provided by services is described. It is also concerned with describing *why* services interact and the capabilities (actions or information) services can provide. Finally, it looks at *how* previously unknown services can “talk” to one another to supply and use the advertised capabilities.

Web service interaction is usually described in the context of a “publish, find, bind” model [64] but this model is too superficial for ad hoc interaction. A more complex model that includes description, advertisement, discovery, evaluation, selection, composition, initiation of dialogue, interaction, and management is required. This thesis is concerned with description, advertisement, discovery, initiation of dialogue, interaction and management. It is not concerned with the issues of evaluation, selection and composition.

The evaluation and selection of discovered services relates to the area of (AI) planning<sup>26</sup>. Planners, following a clients request, use the information given by the data descriptions and service advertisements to find a set of services that are compatible with the clients data model and goals.

The issue of how *client goals* are described and deconstructed into plans is not addressed. The issues involved with evaluating discovered services to establish whether or not they are suitable is also not addressed.

This thesis is not concerned with how services communicate at the transport level. SOAP<sup>27</sup> and HTTP<sup>28</sup> appear to be the protocols with the most acceptance and in this work the existence of transport level protocols that are capable of transporting messages containing the contents described is assumed. The thesis also relies on the existence of services such as UDDI registries for discovering service capability descriptions.

Please note throughout this thesis URI's have been used to identify sources of information, organisations and groups. In all cases the protocol identifier "http://" has been removed to save space. To access any reference in a browser such as Internet Explorer, prefix the given URL with "http://".

## 1.6 Criteria for solution

In the previous section the problems web services encounter when interacting in a heterogeneous environment were outlined. In this section several criteria, based on [153], are identified to be used to judge the quality of solutions to the problems of dynamic ad hoc interaction between electronic services. Solutions should have a *formal* foundation, they should be at a *conceptual* level and have sufficient *expressiveness* to model the problem domain. Solutions must also be *comprehensible* to the people who will use them to create *executable* solutions and they must be *suitable*.

### 1.6.1 Formal

A formal semantics reduces the potential for ambiguity. Both partners in an interaction should be able to understand their options at any point in the interaction process. This is important in the context of computer to computer interaction where formal semantics allows the automated checking of interaction models. The specification of behavioral constraints will also be important in this domain. A formal foundation will ensure these constraints are unambiguous and enforceable.

A formal solution also provides the ability to identify and prove properties of an interaction such as the absence of deadlock and endless-loops, or the presence of appropriate termination conditions.

---

<sup>26</sup>[elektra.mcm.unisg.ch/pbwsc/overview.php](http://elektra.mcm.unisg.ch/pbwsc/overview.php)

<sup>27</sup>[www.w3.org/TR/soap/](http://www.w3.org/TR/soap/)

<sup>28</sup>[www.w3c.org/Protocols/Specs.html](http://www.w3c.org/Protocols/Specs.html)

### 1.6.2 Conceptual

Conceptual level solutions prevent the problem of over-specification. It is important to represent a solution in sufficient detail so that it can be used as the basis for the implementation, but it is also important that the specification does not constrain or dictate which implementation of the solution is appropriate.

A conceptual solution should be easily understood by those who will implement it. It allows mutual understanding of the permissible set of actions and the responses that can be expected. Solutions at a conceptual level do not pre-determine the language or method of an implementation and ensure specific implementation techniques are not imposed on either party.

As an example, many specifications are now formalised in XML schema and this presents problems at the conceptual level. In an XML schema, complex elements that contain sets of data, are often specified as *sequences* of data. In reality, it is the inclusion of the data that is important, but the sequence of its inclusion is not. It should make no difference whether a name is required before an address, as long as both are included. In addition, the sequencing of information belies one of the major advantages of XML; that it is a data description language. Consequently it is sufficient that the data is described as an address (<address>), we do not need to assume it must be an address because it is in the address “position” in the prescribed sequence.

Although XML is emerging as a dominant technology at this time, specifications should be at a higher level so users have the choice of which implementation technology is appropriate to use. This ensures that a solution, if valid, will be useful if and when new technological innovations are made. This is especially important in the web environment where new description languages are emerging such as RDF, RDF schema<sup>29</sup> and OWL.

Solutions to the problem of dynamic two party interactions cannot be entirely the responsibility of one side of the interaction. A solution may require both parties to be at least cognizant of the solution. This does not relax the condition that each party to an interaction must be able to implement the solution in their preferred way.

### 1.6.3 Expressive

An expressive solution allows the modeling of both simple and complex interactions. The language must allow for the basic elements of computation; sequence and choice. It must also be able to represent more complex concepts such as synchronization and concurrency.

An expressive solution will provide a core set of interaction primitives that allow different types of web services to interact within and across domain boundaries. A solution must be able to express variations of behavior depending on the context and competencies of the parties involved. Interaction styles may be domain or

---

<sup>29</sup>[www.w3.org/RDF/](http://www.w3.org/RDF/)

industry specific. For example, purchasing in the food industry is very different to the complexities of the international arms trade. An expressive solution will allow modeling of solutions for many different types of application, from counters and calculators to tax return preparations.

A solution to the problems of dynamic ad hoc interaction should enhance the ability to compose service invocations into sequences of interactions or multi-party interactions. It should allow nested interactions where one party may “sub-contract” parts of the process to other providers.

In a service to service ad hoc interaction context service designers must be able to express constraints on the reactive behavior of participants. What may be allowable in one context may not be in another. A solution must allow for a balance between openness, accessibility and security considerations.

In “traditional” programming, the behavior of software is fully specified in advance by designers and implemented by programmers. In this work it is suggested a software service may need to request a client to perform a variant of behavior that has not been explicitly anticipated by its designer. Under these circumstances it is important the client’s designer can specify limits on which kind of behavioral variations are allowed, or specify the conditions under which variations of a specific type are allowed. For example, the designer should be able to specify “answer questions about the product but do not answer questions about the maximum price we can pay”.

In as much as it is feasible a solution should be fault tolerant and adaptive. This is to reduce the incidence of fatal error conditions which require human intervention. An expressive solution should allow alternative process paths and recovery options.

#### 1.6.4 Comprehensible

A solution must be comprehensible to system designers and service developers to be used effectively. It must be simple enough to be used intuitively and also build on existing solutions using well understood primitives that have a broad basis of support and acceptance. Use of the “Get” and “Set” methods of Object Oriented programming [21] and the REST [59] operations “Get” and “Put” or “Post” will leverage existing expertise.

In many cases comprehensibility issues overlap with the previous criteria. For example solutions at a conceptual level allow comprehension and are easier to share between stakeholders.

A solution should be *easy to use*. The developer of the service provider, or its facade or proxy or broker should not have to expend too much time and effort to create service descriptions and interaction specifications. Solutions that require a large effort may not be used and may also be hard to verify. Ease of use does not imply simplistic, the solution must still be *expressive* enough to describe all the interaction scenarios the solution addresses.

### 1.6.5 Executable

In the context of ad hoc interaction between web services, executability is important. In order to demonstrate that a solution is sufficient and complete enough to describe interactions in the domain of interest, it must be possible to demonstrate that the solution can be implemented, preferably without recourse to proprietary technologies. An implementation where both participants communicate meaningfully in an interaction defined using the provided solution demonstrates that it is feasible and executable.

A solution is executable if it allows services and their clients to interact dynamically to achieve the client's goals. The service provider must be able to gather the information it needs to perform the service and the client must be able to seek clarifying information from the provider. A solution should have "failure" as its last option rather than the first. A solution should minimize the necessity for human involvement by ensuring error handling and error correction can be built in. Executable solutions are needed that can deal with the unexpected and unplanned situations that will characterize dynamic ad hoc interaction.

### 1.6.6 Suitable

A solution to the problem of dynamic ad hoc interaction requires a suitable solution. Solutions may comply with the foregoing criteria but still be unsuitable. For example a solution may be at a conceptual level, sufficiently expressive and executable but still may not be suitable because it cannot be efficiently implemented.

In this context one aspect of suitability is *loose coupling*. Loose coupling ensures the provider who uses the solution to create an interaction plan does not impose requirements on the prior knowledge, description or behavior of the client. The solution must respect the independence of both parties by allowing both participants to behave autonomously within a loosely coupled conversation context. Both parties must have the option to refuse requests for information or functionality.

A solution is *dynamic* if services and clients can interact without clients having to know in advance the sequence of interactions, or the specific requirements of the service provider beyond those contained in the advertisement.

A suitable solution will allow the description of *flexible* and *robust* interaction plans. Flexibility is required to cater for differing client resources and competencies. A robust interaction specification would ensure a successful completion or runtime *error handling* that delivers specific information about the cause of failure. In the heterogeneous environment that will characterize web services in the future, the provision of a form of context sensitive *help* using semantic disambiguation to resolve terminology differences, before a failure is declared, will also contribute to robustness.

A suitable solution should be layered to allow the *separation of concerns* providing the ability to build on existing processes and technologies and the ability to add other layers such as security and authentication when required. A solution to

the problem of dynamic interaction does not stand alone. It stands at the top of a stack of protocols and standards [140] that begins at the transport layer and grows through data representation and formatting to service description and discovery. A solution must work within the constraints imposed by the protocol stack that supports it while retaining the flexibility to adapt to changes in these protocols as they mature.

In many cases security and authentication requirements will be met by the underlying protocols. A solution should also allow clients and providers to layer in their individual security constraints and requirements depending upon their individual operating context. The internal systems of the service provider must be protected from malicious or unwitting clients, just as service clients must be protected from malicious or unscrupulous systems.

It is also desirable for a solution to allow both parties to *monitor* and record the progress of interaction for later verification. This is desirable for several reasons. Monitoring protocol execution appears to be the only effective way of ensuring compliance with formally defined protocols [75, 156, 159]. Monitoring can also be used for security and access control. It is important also in transaction processing for non-repudiation of commitments and enforcing the execution of agreements. Finally, execution logs allow auditors to see how results were achieved and where the problems lie, as part of a process improvement strategy.

The next section introduces a short future scenario that will serve as a running example to illustrate the proposals throughout the thesis.

## 1.7 Case study scenario

Martha heaves a sigh of relief as she gets off the plane in Singapore after a 12 hour flight from London. Martha's flight is a direct flight from London to Sydney so this is just a refuelling stopover not a scheduled stop. As she walks up through the gate lounge her PDA receives an advertisement for an electronics store named GoElectronic in the terminal. She remembers that she has promised to get her son Andrew an MP3 player for his birthday in 3 days.

Martha tells her PDA to make sure she knows when to be back in the gate lounge to re-board the plane. She also tells it to check out the prices of the MP3 players at GoElectronic and give the prices to her in Australian dollars.

The PDA finds a service that provides flight details and tells it to send updates by SMS to Marthas phone. The first message tells Martha her flight is re-boarding in 20 minutes. The PDA checks the online GoElectronic catalogue using the address given in the advertisement.

It finds there is only one MP3 player in stock which costs 323.00 Singapore dollars (SGD). The PDA follows a link in the catalogue to a currency conversion service and tells Martha the MP3 player costs 273.00 Australian dollars (AUD). The PDA checks the airport location guide and it tells the PDA that it would take Martha 15 to 20 minutes to walk from where they are to the GoElectronic store and back again.

Martha decides against walking to GoElectronics and settles instead for a browse in the gift shop she can see from the gate lounge.

## 1.8 Outline

The next chapter (2) is based on work presented at the International Conference on Web Engineering (ICWE'03) held in Oviedo, Spain in July 2003 [122]. In this chapter the WSDL and OWL-S specifications are examined to determine how well they achieve their role as web service interface definition languages.

In chapter 3 an approach to the shared data problem is presented. This chapter is based on work presented at the International Conference on Conceptual Modeling (ER2003) in Chicago, USA in October 2003. A method for developing conceptual models for web-based applications that take into account external definitions and specifications is described. The problem of shared data and data mismatch is addressed by extending traditional conceptual models to include: a way of identifying shared elements; a way to refer to external sources of information (standards and specifications); and a way of referring to other capabilities associated with these elements such as data manipulation or translation. The solution enables service developers to create local type definitions similar to contextualised ontologies. These local definitions can refer to all (or some of) the public standards and specifications that are appropriate in the context of the service. These definitions can also evolve to incorporate new standards as they are developed and to reflect changes in existing standards and the application environment.

Chapter 4 addresses the problem of describing what services can do. It is based on work presented at the International Conference on Service Oriented Computing (IC-SOC'03) in Trento, Italy in December 2003. This chapter shows how Fillmore's case frame can be adapted to provide the basis for a structured description of service capabilities. A structured format allows ad hoc discovery and comparison of services based on combinations of the "cases" contained by the structure and provides a way of ensuring service descriptions are complete or at least sufficient. This service description format provides information about the context of the service, the input parameters it requires and the outputs and/or effects it generates.

To cater for the diverse resources of the potential client base, the capability description allows the description of alternative input sets. For example, a service that converts an XML document to plain text could take as input the document itself or a URI reference to the document, depending on what information the client can

provide (text or URI). The capability description builds upon work in the previous chapter.

In chapter 5 addresses the problem of ad hoc interaction between heterogeneous services. A language for service interaction is introduced. The primary purpose of this language is to enable services as providers to gather the information they require from their clients in order to be able to deliver an advertised capability. Additionally, a means to create interaction plans that direct the dialogue and allow the generation of appropriate messages based on the type of information the service requires is provided. The interpretation and generation of messages in dialogue interactions between services is demonstrated by an implementation in a Coloured Petri Net (CPN).

Chapter 6 presents a summary of the work presented in this thesis, and evaluates the proposals against the criteria introduced in section 1.6.