Chapter 1

A Knowledge-Based Framework for Web Service Adaptation to Context

Carlos Pedrinaci, Pierre Grenon, Stefania Galizia, Alessio Gugliotta, and John Domingue

Abstract. Web services provide a suitable level of abstraction and encapsulation for the development of applications out of reusable and distributed components. However, in order to increase the applicability of services as well as to fine tune their execution in particular situations, context-adaptive solutions are increasingly required. In this chapter we describe a generic knowledge-based framework for supporting the adaptation of Web services and Web service-based applications to context. The framework builds upon research in semantic Web services and Knowledge Engineering to support capturing contextual information in a way that is directly amenable to automated reasoning in order to support the recognition of relevant contextual information that is then used for selecting the most appropriate service or set of services to be executed.

1.1   Introduction

Since the appearance of Web services technologies and impelled by the global interest on Enterprise Application Integration (12) and Service-Oriented Computing (23), there has been much research and development devoted to better supporting the use of Web services as the core constituent of distributed applications. However, despite the appealing characteristics of service-orientation principles and technologies, their application remains mainly limited to large corporations and, effectively, we are still far from

achieving a widespread application of service technologies over the Web. One aspect that is increasingly seen as a *condicio sine qua non* for achieving this is the capacity to dynamically adapt services based on contextual factors (17). These factors range from immediate concerns of location and language to legal issues and financial regulations. Swiftly accommodating to the context at hand will become increasingly important as the diversity of services expands with the global reach of the future Web of services.

Researchers focussing on Web services related technologies are beginning to consider aspects such as security, quality, trust and adaptability within the broader area of context-awareness (13, 16, 17). In fact given the broad meaning of context, characteristics such as low-level execution monitoring data or security and trust concerns are valuable and even necessary contextual information concerning services. Over time many different approaches to developing context-aware applications have been devised (4, 5, 14, 32). In most cases, applications were developed in an ad hoc manner and thus tailored to specific environments, domains, and purposes. As a consequence, from a general perspective the solutions proposed embed a certain set of trade-offs that prevent their systematic application across domains and environments. What can be distilled from this research is the complexity to design, develop and maintain context-aware applications.

These difficulties have motivated the definition of frameworks and architectures for developing and supporting context-aware applications (5, 14, 32). These frameworks, however, need to support adapting services to particular contexts in a systematic manner in order to be adequate to the reality of the Web were one is likely to encounter a virtually infinite number of situations that need to be accommodated. We need, therefore, i) appropriate means for modelling contextual information based on a set of desirable requirements relating to the applicability of an information model, the support for comparing data, and the ability for inferring new data; and ii) a fully-fledged general purpose framework designed to provide an efficient, configurable, robust but nevertheless simple solution to context adaptation.

In this chapter we describe our approach for supporting the adaptation of services to context. We present a framework that is totally based on semantic technologies in order to provide a domain-independent solution. Application developers can use the framework for constructing their context-aware solutions by simply providing a set of domain-specific models that capture the degrees of flexibility the application should have with respect to different contexts and how it should react to changes. In the remainder of the chapter we shall first present an overall view of the framework describing in detail its core components. Additionally, a simple example is introduced in order to better illustrate the notions on which the framework builds upon.

1.2 Web Services Adaptation to Context: Overall approach

The contextual service adaptation framework described herein aims to provide a generic platform for supporting the adaptation of services to diverse contexts according to a virtually infinite variety of dimensions. In fact, our understanding of context is very much in line with perhaps the most widely agreed definition which is presented in (4):

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves."

We here take a slightly wider understanding of context that is not limited to interactions between users and machines but also between machines themselves. After all, one of the main advantages service-oriented technologies provide is the capacity to compose existing services in order to provide more advanced ones.

In the light of this broad understanding of context, our approach to context adaptation is based on a set of conceptualisations providing the means for modelling contextual information of any kind, and a general purpose machinery able to manage contextual information, use it for recognising concrete contexts within particular situations, and apply this contextual knowledge for adapting services. Therefore, central to our approach, much like the definition of context itself, is the genericity of our framework so that it can cover the wide range of situations one is likely to encounter in a Web of billions of services.

Adapting services to particular contexts is envisaged as a process like the one illustrated in Figure 1.1. In a nutshell, this process involves gathering and deriving contextual information, recognising relevant contextual information given a concrete situation, and eventually adapting the execution of the service(s) based on the contextual knowledge gained. Although this process is essentially sequential, while trying to recognise contexts it might be required to acquire more information either from raw data or by deriving additional data on demand, and to trace the information obtained in order to determine how trustworthy it is. We shall not deal with the latter in this chapter.

Our approach to supporting the adaptation of services to contexts is based on previous research in the area of semantic Web services, Knowledge Engineering and Artificial Intelligence for modeling contextual information and for capturing the expertise for supporting Context Recognition and Service Adaptation in a domain-independent way. The kinds of adaptation currently contemplated are i) the use of *late binding* techniques for selecting at runtime the most suitable service to be executed based on contextual information; and
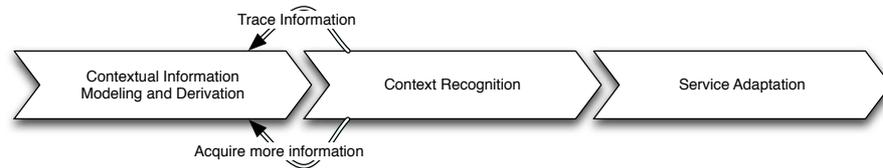
Figure 1.1: Web services adaptation process.

ii) the application of Parametric Design techniques to ensure that processes that use late binding for several activities are analysed such that services are locally contextualised and processes remain globally coherent and optimised. To this end, our framework is composed of three main generic services which support: i) Context Modeling and Derivation; ii) Context Recognition; and iii) Service Adaptation based on Context. In the remainder of this section we shall first describe IRS-III, the execution environment underlying our framework, and will then present the details of each of the services composing the adaptation framework.

1.2.1   IRS-III: a Semantic Execution Environment

Research in semantic Web services aims to increase the level of automation that can be achieved while discovering, selecting, composing, mediating, executing and monitoring Web services. Semantic Web services technologies make use of semantic annotations over existing services in order to support the application of automated reasoning to better support the aforementioned tasks. Several approaches have been proposed so far, the most prominent ones being WSMO (9), OWL-S (18) and WSDL-S (1) which was the basis for the only standard existing nowadays in the field, namely SAWSDL (7).

In our framework we build upon WSMO which provides ontological specifications for the core elements of semantic Web services. WSMO is based on four main building blocks, namely:

1. *Ontologies.* An ontology is an explicit formal shared conceptualization of a domain of discourse (11). Ontologies are used as data models throughout WSMO, so that all resource descriptions and the data exchanged during service execution are formally represented.

2. *Web Services.* Web services represent semantic models of traditional web services with formalised interfaces and capabilities.

3. *Goals.* Goals can be seen both as an abstraction over userÕs goals and as an abstraction over a set of Web Services. Web Services and Goals are the main abstractions relevant for the adaptation of services.

4. *Mediators*. Mediators support a proper integration –through data and process mediation– between any two WSMO elements while they ensure a strong decoupling between them.

Together with the conceptual work around WSMO, much effort has been devoted to implementing what we refer to as Semantic Execution Environments (22, 8). Semantic Execution Environments are systems that can process semantic Web services in order to support the (semi-)automated discovery, selection, composition, mediation, execution and monitoring of services. Work on Semantic Execution Environments is strongly based on WSMO as the conceptual model and is currently under standardisation within OASIS taking as starting point two reference implementations: WSMX (8) and IRS-III (6).

In our framework we build upon IRS-III, a framework for creating and executing semantic Web services, which takes a semantic broker-based approach to mediating between service requesters and service providers (6). IRS-III uses OCML for internal representation and incorporates at its core an OCML reasoner which supports both backward and forward-chaining reasoning (19). OCML is a frame-based language which includes support for defining classes, instances, relations, functions, procedures and rules. IRS-III includes its own implementation of WSMO and a set of core components for handling service discovery, invocation, choreography, orchestration and mediation. The reader is referred to (6) for more details.

A core design principle for IRS-III is to support capability-based invocation. A client sends a request which captures a desired outcome or goal and, using a set of semantic Web service descriptions, IRS-III will: a) discover potentially relevant Web services; b) select the set of Web services which best fit the incoming request; c) mediate any mismatches at the conceptual level; and d) invoke the selected Web services whilst adhering to any data, control flow and Web service invocation constraints. As the reader may realise, automating the transition from Goals to Web Services represents a relatively simple yet powerful technique for ensuring the most suitable solution is used taking contextual information into account. In the remainder of this chapter we shall describe the extensions that have been added to IRS-III in order to support the adaptation of services to contexts, covering context modeling and derivation, context recognition and finally Web services adaptation.

1.3 Context Modeling and Derivation

Context information is not any specific *kind* of information. Rather, for us, it is information that we assume is readily available or extractable from data and that is *contextually relevant for a task*. Thus the stake in context information gathering can be seen as holding onto efficient search space reduction.

This process has to be guided, however, and it is one role of ontologies to help structuring the information space and prepare for readily reduction. It is now widely recognised that context–aware computing can be supported in information systems using ontologies. (28, 2, 15). The role of ontologies in such systems however can be diverse.

We use ontologies for three conceptually separated purposes which are in truth related and complementary in their support to handling contextual information: i) expressing and structuring prospectively relevant information, ii) recollecting the selection criteria for a task, and iii) recollecting contextual information and generating contextual knowledge. These functionalities are supported by the application of problem solving methods in connection with ontological treatment of the relevant aspects feeding into these methods. The second point, in particular, ties into our use of heuristic classification (see section 1.4.1). Our framework handles these aspects modularly with i) ontologies having the general purpose of dealing with Web services (WSMO), temporal aspects and knowledge of that level, ii) dimensional ontologies dealing with quantities, units and their conversions, and other related aspects, and iii) a context modelling ontology dedicated to supporting ontological treatment of context and contextualistion. In the reminder of this section we describe the main ontological constructs needed to support our framework.
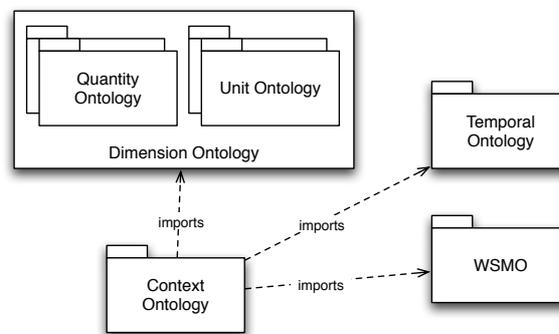

Figure 1.2: Modular ontologies each dedicated to different aspects and recollected within the context modelling ontology.

1.3.1   Background knowledge representation

Our system rests on knowledge–based representation of information about entities showing in the execution of web services. We thus apply generic on-

tological engineering techniques using OCML (19) as our representation language. As mentioned earlier, OCML is a frame–based language and it is used, typically, to define collective structures, *classes*, bearing slots (attributes) allowing to characterise *instances* of classes by associating values to slots. Thus in laying out the information about entities of interest (users, services and so on) we rely primarily on the slots & attribute-values mechanism. Against this background we focus on knowledge that can be collected by associating instances (for example, a particular user) with certain values (for example, an age on a given scale). Values in this sense are, of course, instances of certain classes and this forms the basis of our notion of dimension.

Our dimension ontology, shown in Figure 1.3, has at its core the class *dimension*. Figure 1.2 sketches the ontology in abstraction of its modularisation. There are different kinds of dimensions and in our dedicated ontology the class *dimension* has a number of specialisations. We can describe the typology of dimensions according to two main aspects: i) structural and ii) logical features. Structural aspects have to do with characteristics of the values that are members of dimensions. The most significant structural kinds for our purpose are, firstly, those of dimensions whose member values can be ordered and, secondly, those of dimensions whose member values are given in units. The definition of an ordered dimension is completed by the association of the dimension with an order relation that holds between its members. Because of this, a set of values can be common to two dimensions while the dimensions will remain distinct if the values are differently ordered. The definition of a dimension with values being given in units is less direct. Because dimensions are defined, partially, by a set of values, we define dimensional values *in abstracto* and independently of dimensions. The only thing we give to every dimensional value is a magnitude. We define a specialisation of *dimension value* that is instantiated by values whose magnitude is only meaningful in a unit. From such assemblies of values we can then define dimensions with units.

Thus, an instance of *dimension* is defined by: i) a number of (dimensional) values and ii) a slot that attaches these values to an entity[1]. The values in the dimension can be declared to belong in the dimension explicitly or they can be inferred when a test exists. Figure 1.4 shows an OCML listing defining the class *dimension* and examples of defined dimension for levels of trust. We can define two dimensions *LMH* and *MH* whose values are values of level of trust (for the sake of simplicity we ignore the associated slot). *LMH* and *MH* differ in that the former encompasses all levels of trusts that have been defined (hence we use as a membership test the unary-relation that holds of every levels of trust) and the latter is a more selective one for which the values are declared to include medium and high levels of trust only.

---

[1]The definition of a dimension 'mentions' the slot but it is a slot that characterises a class to which belong entities to be prospectively contextualised.
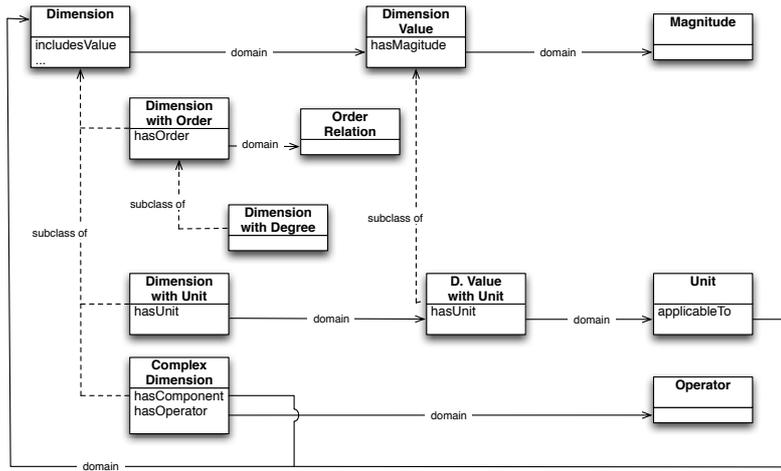
Figure 1.3: Main classes and slots with their domains in our dimension ontology—modularisation is not represented.

```
(def-class dimension ()
  ((dimension-includes-value :type dimension-value)
   (associated-slot :type slot :max-cardinality 1)
   (value-membership-test unary-relation)))


(def-class trust-dimension (dimension))


(def-class trust-value (dimension-value) ?x
  ((value-in-dimension-type :value trust-dimension))
  :iff-def (value-in-dimension-type ?x trust-dimension))
```

```
(def-instance high-trust trust-value
  ((has-magnitude high-degree)))

(def-instance medium-trust trust-value
  ((has-magnitude medium-degree)))

(def-instance low-trust trust-value
  ((has-magnitude low-degree)))

(def-instance LMH-trust-dimension trust-dimension
  ((value-membership-test trust-value)))

(def-instance MH-trust-dimension trust-dimension
  ((dimension-includes-value medium-trust high-trust)))
```

Figure 1.4: Examples of OCML definitions of dimensions.

The second aspect allowing to distinguish among dimensions is the logical one. The logical features involved here are those that have to do with the relative complexity and the relative convertibility of dimensions. By relative complexity of dimensions we refer to cases in which a dimension's values are obtained as a result of applying an *operator* to two or more *dimensions*. The operands are values in dimensions that are simple in relation to the dimension to which belong the result of the applied operator. A simple example of such operation is mere conjunction of two dimensions in which the values in the complex dimension are defined as conjunctions of values in two simpler dimension. Thus, for example, *security* could be seen as a complex dimension if a value in that dimension corresponded to a value in an *encryption level* dimension joint to a value in a *certification reputation* dimension. Convertibility is a somewhat similar affair but is between two dimensions with values given each in particular units such that there exist a conversion operation between these units (e.g., meters and inches). Figure 1.5 gives a simple illustration of a complex dimension of hit rates based on the combination of numbers of hits and time periods.

```
(def-class complex-dimension (dimension) ?x          (def-instance hit-rate-dim-1
  ((has-component-dimension :type dimension)           dimension-with-unit
   (has-dimension-operator :type dimension-operator))    ((has-component-dimension hit-number-dim-1
  :iff-def (exists ?y (has-component-dimension ?x ?y)))                              time-span-dim-1)
                                                       (has-dimension-operator divide)
(def-class dimension-operator (function) ?x)           (has-unit hit-day)))

(def-instance hit-number-dim-1 dimension             (def-instance 10-hit-week dimension-value
  ((dimension-includes-value 10-hit 100-hit 1000-hit)))  ((has-magnitude 10/7)
                                                        (has-unit hit-day)
(def-instance time-span-dim-1 dimension-with-unit       (value-in-dimension hit-rate-dim-1)))
  ((dimension-includes-value 1-day 1-week 1-month)))
```

Figure 1.5: The hit rate dimension is defined as a complex dimension combining two simpler ones.

So far, elements of our framework is reminiscent of the approach taken in (29) which also ties objects of interests to certain attributes using a notion of *aspect* to which resemble our notion of *dimension*. Our approach is somewhat finer grain insofar as it allows to build dimensions from sets of values while CoOL with its ASC model starts from what could be called in our terms generic dimension types and is preoccupied with convertibility between dimensions with units. This should be taken with a grain of salt and as an helper to intuition as we have not tried to establish the precise correspondence as we think it is not a direct one. Another difference with CoOL which is more significant is that while it is designed as a language for representing contextual information, it leaves open ways in which contextual knowledge is

to be selected. We now provide the elements of a needed mechanism which is tied intimately with our treatment of dimensions.

1.3.2   Context reduction

Our context gathering mechanism relies on an ontological treatment of context information reduction to context knowledge – any contextual information that is relevant to the task at hand. We use a dedicated ontology for specifying the required elements that extends all that we need for background knowledge representation (Figure 1.2), namely the dimension ontologies where reside the elements just described but also WSMO which we use to model Web services.

We can contextualise any entity of interest in a systematic and uniform way. To do this, for any entity to be contextualised, we define a *context reduction* associated with this entity which, possibly after a number of processing steps, is able to recollect all the contextual knowledge relevant to the task at hand. While a context reduction is associated with only one entity, a contextualised entity may have different context reductions with which it is associated. Thus, we can also allow for navigating seamlessly between contexts.

The main element in our context ontology is the class of *context-reduction* of which an instance is an artifact allowing to recollect contextual information for an entity to be contextualised and to tie this information to the background activities triggering the delineation of the context in question (tasks and putative goals, in particular). Here we represent this tie as part of the definition of an instance of context reduction but for the sake of generality we could have used a relation rather than a slot.

```
(def-class context-reduction
   (time:TimeSpanningEntity)
   ((context-reduction-of :type ocml-thing)
    (context-reduction-in-dimension :type dimension)
    (has-context-value :type dimension-value)        ; individual values
    (has-context-value-set :type set)                ; collecting set
    (has-associated-goal :type goal)                 ;  wsmo goal
```

Figure 1.6: OCML listing of the definition of the class *context-reduction.*

The slot context-reduction-of associates a context-reduction to an entity to be contextualsied by it. The slot context-reduction-in-dimension associates a *context-reduction* to *dimensions* in which values representing contextual information will be selected to generate contextual knowledge. The slot has context-value and its -set variant are collection slots for gathering contextual knowledge in the form of relevant values. As can be seen from the first line,

context–reduction is a specialisation of a class in the temporal ontology (25) that allows to associate context-reduction with a time–span during which the reduction is considered valid. In our implementation we use a ternary relation in order to relate a contextualised entity (e.g., a user, a Goal, or a Web Service) to values of interest in a given context. This relation is populated through a call to a LISP function taking a context–reduction as an argument and computing the required values of the other arguments of the relation. It is this collected information, which provides the ground for applying context–parameterised heuristic classification.

1.4   Context Recognition

We understand context recognition as a knowledge intensive task that, given a body of information capturing contextual information about services and users, and a particular purpose for which a context has to be identified, returns the concrete context recognised. Our epistemological basis for context information derivation and context recognition is based on Heuristic Classification, a Knowledge Level (21) method that "relates data to a pre-enumerated set of solutions by abstraction, heuristic association, and refinement" (3), see Figure 1.7. In the remainder of this section we shall first present Heuristic Classification and then describe how we apply this method for context recognition.

1.4.1   Heuristic Classification

Heuristic Classification is a particular technique for classifying that was identified, thoroughly analysed and extensively described by Clancey in his seminal paper (3), (see Figure 1.7). For instance diagnosing diseases has often been approached in Artificial Intelligence as a process whereby basic observations about patients like temperature (e.g., 39ij C) are abstracted (e.g., temperature is high), matched against prototypical diseases (e.g., the flu), and refined so that the concrete diagnosis is the one that best explains the symptoms observed. Indeed, the previous scenario has been kept simple for clarity purposes. Realistic scenarios include a wide range of observations (e.g., fever, vomiting), which may have causal relations (e.g., high temperature might cause a certain disorientation), and all need to be reconciled to obtain a correct classification (e.g., fever and vomiting because the patient has gastroenteritis).

In most of the situations, the solution features will not be directly provided by the data observed. Instead they need to be abstracted from the raw data that is available. There are three different kinds of abstraction:
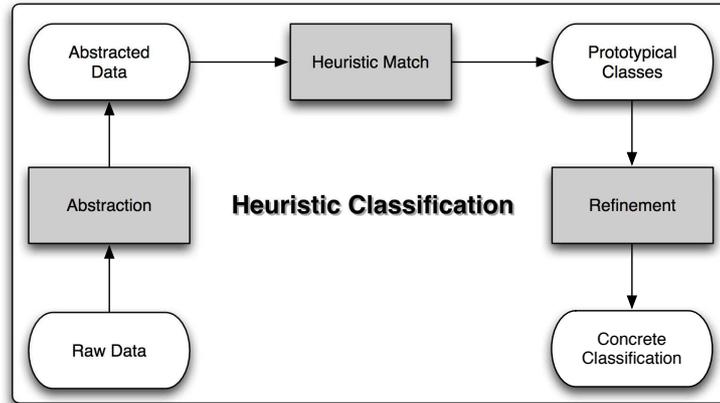
Figure 1.7: Heuristic Classification.

Definitional abstraction is based on the essential characteristics of the concept being analysed (e.g., a computer scientist knows about computers).

Qualitative abstraction is the derivation of some qualitative feature based on quantitative measures (e.g., if service A takes longer than 20 seconds it is performing slowly)

Generalization is based on the use of hierarchical information (e.g., a man is a mammal)

Refinement is a similar process but the direction of the inference is inverted. The goal is to, once a prototypical class has been identified, refine the solution trying to explain most of the observed facts. It is worth noting that refinement takes place within the target classification ontology trying to explain all the facts observed which are typically expressed in terms of a different data model.

The most distinctive feature of Heuristic Classification is the fact that a heuristic is used for achieving a non-hierarchical direct association between the abstracted data and the prototypical classes contemplated. The knowledge for relating problem situations to solutions is thus essentially experiential. A heuristic relation is uncertain, based on certain assumptions of commonality, and allows a reduction in search by skipping over intermediate relations. Heuristic Classification does not attempt to generate a solution that has not previously been identified and therefore, should the prototypical classes be too vague or the scope of the domain they capture too narrow, the solution obtained will also be affected.

1.4.2 Context Recognition as Heuristic Classification

In our framework we approach the task of determining a relevant context based on a task we want to achieve and a wide range of information concerning users and services, as a process involving the abstraction of information gathered, its heuristic classification with respect to general cases, and the refinement to the particular case at hand. In fact, it is this very process of heuristically classifying contextual information that produces contextual knowledge which can then support the appropriate adaptation of services. The reader should notice that we are here distinguishing between contextual information (i.e., all the contextual information gathered may it be useful in the situation at hand or not) and contextual knowledge which can support the system acting rationally (21).

This process is supported in our framework by a library of Heuristic Classification Problem-Solving Methods (PSM) developed by Motta et al (20). In a nutshell, PSMs are basically software components that encode sequences of inference steps for solving particular tasks in a domain-independent manner so that they can be applied for solving the same task within different domains. The library we use is based on the Task Method Domain Application (TMDA) framework (19). TMDA prescribes constructing Knowledge-Based Systems based on the definition of task ontologies that define classes of applications (e.g., diagnosis, classification), method ontologies that capture the knowledge requirements for specific methods (e.g., heuristic classification), domain ontologies that provide reusable task-independent models, and application ontologies for mapping domain models with domain-independent problem solvers.

The library defined in (20) includes a task ontology and a method ontology. Classification ontology characterises the task as the problem of finding the solution (a concept) which best explains a certain set of facts (*observables*) about some individual, according to some criterion. The concept *observables* refers to the known facts about the object (or event, or phenomenon) that we want to classify. Each *observable* is characterized as a pair of the form *(f, v)*, where f is a feature of the unknown object and v is its value. Feature is anything which can be used to characterize an object, such as a characteristic which can be directly observed, or derived by inference. As is common when characterizing classification problems–see, e.g., (31), we assume that each feature of an *observable* can only have one value.

The solution space specifies a set of predefined classes (*solutions*) under which an unknown object may fall. A *solution* itself can be described as a finite set of feature specifications, which is a pair of the form *(f, c)*, where f is a feature and c specifies a condition on the values that the feature can take. Thus, we can say that an *observable (f, v)* matches a feature specification *(f, c)* if v satisfies the condition c. The current implementation contemplates different solution criteria. For instance, we may accept any solution, which explains some data and is not inconsistent with any data. This criterion
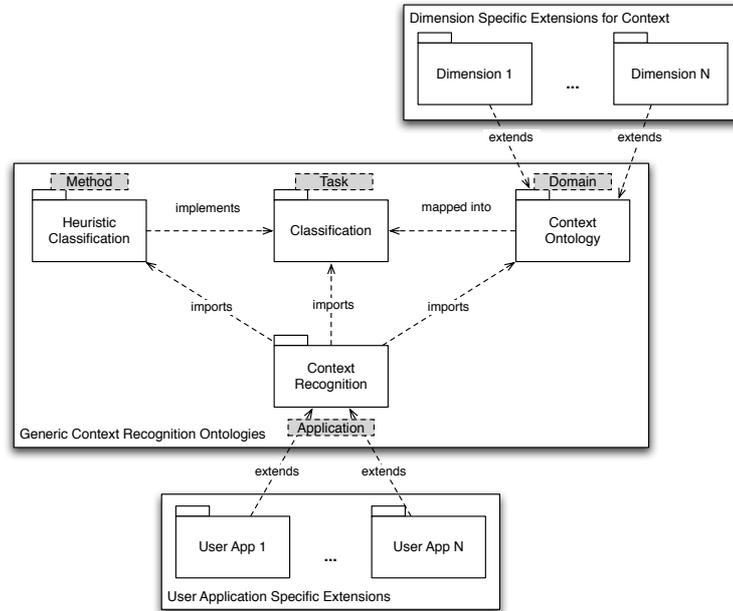
Figure 1.8: Context recognition ontology stack.

is called positive coverage (27). Alternatively, we may require a complete coverage - i.e., a solution is acceptable if and only if it explains all data and is not inconsistent with any data. Thus, the specification of a particular classification task to be performed needs to include a solution (admissibility) *criterion*. This in turn relies on a *match criterion*, i.e., a way of measuring the degree of matching between candidate *solutions* and a set of *observables*.

The library approaches classification as a data-driven search where *observables* try to be explained in accordance to some given criteria. Our classification library implements two different classification methods: *single-solution-classification*, and *optimal-classification*. The former implements a hill climbing algorithm with backtracking to find a suitable solution; the latter executes an exhaustive search for an optimal solution. Abstraction is supported by so-called *abstractors* which are basically functions that given a set of *observables* produce additional *observables*. Conversely, refinement is supported by so-called *refiners* which can be seen as the inverse of *abstractors*. *Refiners* themselves are also defined as functions that given the solution space refine it towards the solution. Coming back to our previous discussion concerning context derivation, it is quite clear that the context derivation techniques (metrics computation, derivation of values for composite dimensions, etc) are *abstractors* within our framework.

In order to use it for our particular purposes, i.e., recognising contexts, it is

necessary to provide a concrete domain ontology, and an application ontology that performs the appropriate mappings between task concepts and domain concepts, and provides heuristic methods where appropriate, see Figure 1.8. Context Ontology (see Section 1.3) is provided as a general purpose domain ontology which users can extend in order to include particular dimensions of interest. Additionally, we provide as part of the framework a general purpose application ontology, namely Context Recognition ontology, which provides mappings between Context ontology and Classification ontology. In particular, the ontology maps the notion of *dimension-value* to the notion of *observable*, and the notion of *operator* to that of *abstractor*, see Section 1.3. In this way applying the framework for concrete applications only requires the provisioning of user-defined dimensions (or typical ones), the set of prototypical classes we want to match our context to, and domain-specific heuristics.

1.5   Web Service Adaptation

Once a relevant context or sets of contexts have been identified, services need to be adapted accordingly. The last infrastructural service provided by the framework is precisely in charge of this. Our framework provides the capability to adapt to particular contexts through two main complementary means. On the one hand we make use of the notion of *late binding* as supported by Semantic Execution Environments like IRS-III or WSMX (22). On the other hand, we enrich this feature with the capability to configure processes taking into account the possible adaptations of all the internal activities in order to maintain their compatibility and optimize the resulting compositions according to different criteria.

1.5.1   Goal-based Adaptation of Services

Like in real life, Goals can most often be achieved in several ways. The "best" one and even those that are suitable depend on a variety of aspects, many of which are merely contextual. Imagine for instance that we want to pay a certain object in an Internet shop. This could be achieved using different kinds of credit cards or even systems like Paypal. The suitability will depend on the kind of cards we have available, on whether we have a Paypal account, on the systems that we trust, etc. The "best" solution for us will depend on our preferences in terms of cost, security, simplicity, etc. These factors are all contextual since the essence of the service, that is to pay a certain amount of money, remains unchanged.

Our framework supports what we refer to as Goal-based adaptation of services by leveraging the capability-based invocation feature provided by IRS-III

(see Section 1.2). This capacity for selecting the service to be executed at runtime is what is often referred to as *late binding*. Late binding support brings manageability to processes by specifying sets of suitable services by means of Goals (24). Additionally, this feature supports deferring the selection of the service to be utilised until runtime where concrete contextual information regarding the services (e.g., their latest performance, availability, price, etc) can be taken into account.

So far the selection of suitable services in IRS-III was uniquely driven by the functionality provided. As a consequence services providing compatible functionalities as specified in their capabilities were all considered suitable in no particular order (see (6, 9) for more details). There are however many non-functional properties (e.g., price, quality of service, trust) which should also be taken into account for fine-tuning the selection of services at runtime. IRS-III has been extended towards supporting context-adaptive capability-based invocation of services by taking into account contextual knowledge in order to restrict and rank the set of suitable services, and eventually choose the most appropriate one.

Somewhat implicit in the definition of context we introduced at the beginning of the chapter, is the fact that the relevance of contextual information is dependent on the actual application or service being provided. For instance, the encryption system used may well be irrelevant for a weather forecast service, whereas it is very important for payment services for example. The underlying essence is that what can be considered as relevant contextual information depends on the task being performed. In a scenario where a large number of services are provided and consumed by billions of users over the Web, the diversity of the tasks that will be supported will presumably be outstanding and there needs to be appropriate means for supporting adapting services to specific contexts in a generic yet scalable manner.

In order to cater for a context-adaptive execution of services we include in Goals a non-functional property (*contextually-sensitive-to*) specific to our framework. This non-functional property identifies context *dimensions* for which the Goal presents a certain *sensitivity* or, if you like, *flexibility*. In essence it identifies a set of context *dimensions* that can affect the late binding process. At runtime the framework can identify these degrees of flexibility, and use them for guiding a context reduction process whereby context knowledge is constructed out of the existing body of contextual information. Once this contextual knowledge is obtained, a Heuristic Classification problem-specification is generated and solved using the mechanisms previously introduced. During this process, an additional classification of services according to the contextual dimensions and heuristic methods are also necessary. The solution to this problem, which as we saw earlier can be restricted to the one that better fits the situation at hand, identifies the kinds of services that should be used for invocation, therefore bringing context-aware adaptability to the invocation of services.

1.5.2 Adaptation of Processes

Goals provide us with the required level of abstraction able to introduce contextual aspects within the execution of services. In simple cases, like the simple one outlined above, adapting services to a certain context is a matter of introducing within the execution infrastructure the capability to recognise a relevant context and use this information to restrict the suitable services and select the best option. In more complex cases like processes, which we define as orchestrations of Goals (6), contextual factors of one Goal may have implications over others. Imagine for instance that we want to contract an Internet connection and a VoIP solution with a certain quality of service (QoS). The kind of connection chosen might not support the QoS desired, yet an Internet connection must be in place before contracting the VoIP solution. Supporting this simple process in a completely automated way reveals to be more complex since context changes provoked by the first Goal (i.e., get an Internet connection) might prevent us from achieving the second Goal (i.e., get a VoIP with a certain QoS). Indeed, it is quite easy to envisage situations where the interdependencies are such that the complexity of the problem cannot be neglected.

In order to deal with this kind of situations and still maintain a level of flexibility allowing us to adapt the execution of processes and services to contexts, we envisage approaching this problem as a Parametric Design problem. This approach is similar to that of (30) although we aim to provide a general purpose infrastructural solution that will allow Semantic Execution Environments like the IRS-III or WSMX (8, 22) to execute contextualised semantic Web services in a generic way.

Parametric Design is a simplification of Configuration Design (19, 26), whereby the objects to be configured are assumed to have the same overall structure as preconfigured templates. For example, when configuring a computer, experts know that there will be a processor, a motherboard, a hard drive, etc. In these cases, design problem-solving consists of assigning values to the parameters defined in the preconfigured template taking into account a set of needs, constraints, and desires. This kind of problems, greatly decrease the complexity of the problem-solving task with respect to other types of designing and there exist tractable implementations (19).

One of the most comprehensive toolsets for solving Parametric Design problems was formalised and developed on top of the TMDA framework previously introduced (19). Therein Parametric Design is defined as a task that given a set of *parameters*, a set of *constraints*, a set of *requirements*, a *cost function*, a *cost algebra*, and a set of *preferences*, obtains a *design model*. *Design models*, that is the solutions to a Parametric Design problem, are defined as a set of parameters assignments of the form *(parameter, value)*. *Constraints* specify conditions that need to be satisfied by a design, and therefore restrict the solution space. *Requirements* on the other hand describe desired properties solutions should have. Requirements are therefore the positive counterpart

of *constraints*. *Preferences* are basically the means for ranking the different solutions obtained. They are therefore different from *requirements* and *constraints* since they do not restrict the space of possible solutions but rather establish an order between the solutions obtained. Finally, the *cost function* and the *cost algebra* simply support the introduction of a global preference system for ordering solutions based on the combination of all the preferences provided.

In addition to the generic task definition, (19) includes a set of PSMs for parametric design. The library includes a general purpose search-based PSM based on the notions of *design state* and *design operators*. In this case Parametric Design is approached as the exploration of the possible design states which can be reached by applying the existing *design operators*. More advanced problem-solving strategies such as Propose & Backtrack or Propose & Revise are also supported in order to achieve further performance by including additional knowledge for guiding the exploration of the solution space. The trade-off indeed lays on the fact that more advanced problem-solving techniques require additional knowledge.

The adaptation of processes through Parametric Design is achieved by considering the overall control structure of the Goals orchestration as the preconfigured template. The Goals composing the orchestration represent the *parameters* and each of them can take a set of Web Services as value. Additional *constraints* or *preferences* between the different *dimensions* to which each of Goals are sensitive to can be specified so as to restrict unsuitable services combinations or to favour certain solutions over others. Solving the Parametric Design problem is a matter of selecting services (that is assigning values to the *parameters*) such that the *constraints* and *preferences* are honoured. The result obtained is an orchestration of services which has been adapted to context based on a set of constraints and preferences.

1.6   Application

So far we have presented the framework from a Knowledge-Level perspective focussing on the kinds of knowledge that are relevant and how they can be utilised for adapting services to context. In this section we present a simple application that aims to illustrate the notions presented so far. In this example, however, we shall limit ourselves to one single adaptation mechanism, namely Goal-based adaptation, for simplicity and space reasons.

The application is centred around the notion of *trust* which is indeed a very important aspect regarding services and their execution. The main issue of representing trust in all its facets lies in its contextual nature. Different users do not necessarily agree on which services are trustworthy. For example, a user

may trust a Web service with a highly rated security certificate whenever she has to provide her credit card details, but may prefer an accurate service when precision is more important than security. Conversely, the same user weights the opinions of past users about a specific Web service in other situations, i.e., the evaluation of the Web service reputation is a priority in the current trust understanding of the user. Finally, distinct users may privilege different trust parameters in the same context; in this case, their priorities may depend on their different personal preferences.

Therefore, a trust-based mechanism for selecting Web services represents a compelling application of our framework. Following the principles introduced in the previous sections, we introduce an abstract model for trust that enables interacting participants–both Web service users and providers–to represent and utilize their own trust understanding with a high level of flexibility, and thus take the possible multiple interacting contexts into account.

1.6.1   The Trust Framework

Adapting service selection based on how trustworthy services are is here understood as a two steps process. Firstly, all participants specify their own requirements and guarantees about a set of trust parameters. Then, at runtime, our objective is to identify the class of Web services that matches the trust statements of involved participants, according to an established classification criterion; i.e. achieving Web service adaptation on the basis of trust-related parameters, see Section 1.5. In order to apply our framework to support the selection of services based on trust–and thus verify its benefits–we have extended the context model with trust-specific dimensions based on our previous work on Web Services Trust-management Ontology (WSTO) (10).

User preferences are the main elements on which Web service selection depends. Essentially, the user can decide which parameters should be considered in order to determine which class of Web services are trusted, in a given context. Therefore, in WSTO, the concepts *user*, *Web Service* and *Goal* are central ones, whereby *user* denotes the service requester and *Web Service* the service provider. Following the basic WSMO notions, a *Goal* represents the service requester's desire or intention. The user usually expresses different trust requirements in achieving different goals. For example, she can be interested in data accuracy when retrieving timetable information, and security issues when disclosing her bank accounts. On the other hand, the service aims to provide a set of trustworthy statements, in order to reassure the requester as well as to appear as attractive as possible.

The participants (*Web service* and *user*) are associated with trust profiles, represented in WSTO by the class *trust-participant-profile*. A profile is composed of a set of trust requirements and guarantees. Both *trust-guarantees* and *trust-requirements* are values within some dimension (e.g., *encryption-mechanism-dimension*), however their treatment within the adaptation process is different. Trust-guarantees are observables, i.e., contextual information

that is available in the system, whereas *trust-requirements* are instead candidate solutions (pairs of feature and condition *(f, c)*).

We distinguish three logical elements in trust requirements: (i) a set of candidate solutions for expressing conditions on guarantees promised by the relevant parties; (ii) a candidate solution for requesting their reliability; and, (iii) a candidate solution for requesting their reputation evaluation. In a participant profile, the three elements are optional; choice depends strictly on the participant preferences in matter of trust. Similarly, the participant trust guarantees have three components: (i) a set of observables for representing the promised trust guarantees; (ii) an observable corresponding to the evaluation of the participant reliability; and, (iii) an observable for representing the reputation level of the participant. Whereas the promised trust guarantees are a set of promised values stated by the participant–such as (*execution-time, 0.9*) and (*data-freshness, 0.8*)–reliability and reputation guarantees are computed automatically based on data captured by monitoring software.

WSTO is comprehensive of all approaches usually adopted for representing trust (10). It embeds a policy-based trust management since the interacting participants express their trust policies in their semantically described profiles. The Semantic Execution Environment—in our case IRS-III—will behave as a Trusted Third Party (TTP) by storing participant profiles and reasoning on them. Moreover, the reputation module enables a Web service selection also based on ontological statements about reputation.

1.6.2   Case Study: A Trusted Virtual Travel Agent

The current prototype considers participant observables and needs, but it does not include the reputation module and the historical monitoring. As an illustration we implemented a simple virtual travel agent service whose goal is to find the train timetable, at any date, between two European cities. Origin and destination cities have to belong to the same country (European countries involved in our prototype are: *Germany*, *Austria*, *France* and *England*).

The client using this application publishes her *trust-profile*, with trust requirements and/or trust guarantees. In our prototype, we provide three different user profiles and three different Web services able to satisfy the user's goal, following the lines of Section 1.3 for the representation of knowledge. User profiles are expressed through trust requirements, without trust guarantees. All user requirements are performed in terms of security parameters: *encryption-algorithm*, *certification-authority* and *certification-authority-country*. These security parameters are three examples of possible *dimensions* for the trust domain. The listing of Figure 1.9 illustrates how users express a qualitative level of preference for each parameter.

For instance, *user4* would like to interact with a Web service that provides a high security level in terms of encryption algorithm, but she accepts medium value for certification authority and certification authority country. User requirements are represented in a qualitative way which is a more con-

```
USER4                                                    USER5
(def-class trust-profile-USER4                           ...
  (trust-profile)                                        (def-instance requirement-USER5
  ((has-trust-guarantee :value guarantee-USER4)            contextualised-trust-requirement
   (has-trust-requirement :value requirement-USER4)))      ((has-encryption-algorithm medium-trust-req)
                                                            (has-certification-authority low-trust-req)
(def-instance requirement-USER4                             (has-certification-authority-country low-trust-req)))
  contextualised-trust-requirement
  ((has-encryption-algorithm high-trust-req)
   (has-certification-authority medium-trust-req)        USER6
   (has-certification-authority-country medium-trust-req)))  ...
                                                         (def-instance requirement-USER6
(def-instance user4-requirements-reduction                 contextualised-trust-requirement
  context-reduction                                        ((has-encryption-algorithm low-trust-req)
  ((context-reduction-of requirement-USER4)                 (has-certification-authority high-trust-req)
   (context-reduction-in-dimension                          (has-certification-authority-country high-trust-req)))
     required-datafreshness
     required-certification-authority
     required-certification-authority-country)
   (has-associate-goal get-train-timetable)))
```

Figure 1.9: OCML listing of examples of user profiles and preferences.

venient way for users. As explained earlier, Heuristic Classification relies on an abstraction step that takes raw data and transforms it into higher-level information. The listing in Figure 1.10 shows an example of a particular abstractor that can take low-level information about the encryption algorithm used and transform it into a qualitative representation of its security[2].

```
(def-instance encryption-algorithm-abstractor
  abstractor
  ((has-body '(lambda (?obs)
   (in-environment ((?v . (observables-feature-value ?obs 'has-encryption-algorithm)))
   (cond
     ((== ?v DES) (list-of 'has-encryption-algorithm 'high-trust-req (list-of (list-of 'has-encryption-algorithm ?v))))
     ((== ?v AES) (list-of 'has-encryption-algorithm 'medium-trust-req (list-of (list-of 'has-encryption-algorithm ?v))))
     ((== ?v RSA) (list-of 'has-encryption-algorithm 'low-trust-req (list-of (list-of 'has-encryption-algorithm ?v)))))))))
```

Figure 1.10: Encryption-Algorithm Heuristic OCML listing.

The heuristic *encryption-algorithm-abstractor* establishes that whenever the encryption algorithm adopted by a Web service provider is like *DES*, then its security level is considered high-trust-req. Whenever both User and Web service describe their profiles, they implicitly agree with the qualitative evaluation expressed in the heuristic. In turn, whenever the Web service provider makes use of an algorithm like *AES*, according to the heuristic in Figure 1.10, its encryption ability is deemed medium-trust-req, otherwise, if the adopted algorithm is like *RSA*, the security level is low-trust-req.

Other heuristics provide qualitative evaluations of *Certification Authorities (CA)*, and *CA countries*. For instance, security level of *globalsign-austria* is

---

[2]The categorisation we use is arbitrary and for the sake of illustration

Figure 1.11: Web application.

retained high, conversely German CAs are considered medium-secure[3]. The user can apply these heuristics, or define her own, sharing her expertise and knowledge with other users. Alternatively, the user can even express her requirements in a precise/quantitative way, by specifying the exact values expected from Web service guarantees, for example, the certification authority issuing security token has to be *VeriSign*.

We developed a user-friendly Web application to test our implementation, which is available at http://lhdl.open.ac.uk:8080/trusted-travel/trusted-query. The snapshot in Figure 1.11 shows the Web application interface. The user who would like to know the timetable for trains between two European cities enters the desired city names and date. The user owns a trust profile associated to her name: e.g. *dinar* is for instance associated to *user4*, *vanessa* to *user5* and *stefania* to *user6*.

Whenever the application starts, the system identifies the trust-profile based on the user name. Given several Web services semantically described in terms of WSMO, all with the same capability, but different trust profiles, the application is able to select the most trust-worthy service. Indeed, given the *trust-guarantees*, i.e., the *observables*, and the *trust-requirements*, i.e., the *candidate solutions*, selecting services based on trust is simply a matter of running the adaptation mechanism provided by the framework. For this application we applied the complete coverage criterion, i.e., every user requirement is explained (matches with a Web service trust guarantee) and none is inconsistent, and we used the optimal classification method which ensures the Web service selected is the most trustworthy one.

---

[3]Again, the reader should note that this simply arbitrary decisions

Trusted Travel


```
{
 Applicable Web Services:

GET-TRAIN-TIMETABLE-SERVICE-T3

GET-TRAIN-TIMETABLE-SERVICE-T2

GET-TRAIN-TIMETABLE-SERVICE-T1


 The WS class that matches with DINAR trust requirements is : GET-TRAIN-TIMETABLE-SERVICE-T1


The result is:

 "Timetable of trains from Berlin to Frankfurt on 18, December, 2008
6:47
7:35
8:23
9:11
9:59
10:47
11:35
12:23
13:11
13:59
```

Figure 1.12: Web Application Output for the user "dinar".

Figure 1.12 is a snapshot of the resulted trusted VTA booking. The application returns the list of Web services able to satisfy the user goal, and the one that is invoked, which is the Web service that matches dinar trust requirements. It follows the Web service output, the requested timetable. Easily, the application can be tested with the other user instances implemented, *vanessa* and *stefania*. It can be noticed that *vanessa* trust profile matches with Web service class *get-train-timetable-service-T3*, while *stefania* with *get-train-timetable-service-T2*. A ""non-trusted" based version of the application is available at http://lhdl.open.ac.uk:8080/trusted-travel/untrusted-query for comparison purposes. This application implements a virtual travel agent based on the standard IRS-III goal invocation method. The output returns only the train timetable requested, without any trust based selection.

1.7 Conclusions

Service-Oriented Architecture is commonly lauded as a silver bullet for Enterprise Application Integration, inter-organizational business processes implementation, and even as a general solution for the development of all com-

plex applications. However, despite the appealing characteristics of service-orientation principles and technologies, we are still far from achieving a widespread adoption of service technologies over the Web as initially predicted.

One aspect that is increasingly regarded as one of the main limitations of service technologies is the fact that current technologies do not support adapting services and processes behaviour to context. Gradually researchers focussing on Web services related technologies are beginning to consider aspects such as security, quality, trust and adaptability within the broader area of context-awareness. Even though appealing results have been achieved so far, most of the solutions created are rather ad hoc and can hardly cope with the road range of heterogeneous cases one could find should service technologies be widely adopted in the Web.

In this chapter we have presented a knowledge-based approach to adapting services in a generic automated and effective manner. The framework described is based on research on semantic Web services and Knowledge Engineering and provides a number of domain-independent services that can take contextual information and use it for supporting the adaptation of services and processes. The framework uses ontologies as its core backbone for context modeling and reasoning. Additionally it relies on its capacity for recognising relevant contexts by using Heuristic Classification and for adapting processes by configuring orchestrations of Goals using a Parametric Design engine.

The main advantage of the framework is that application developers can directly use it for constructing their context-aware solutions by simply providing a set of domain-specific models that capture the degrees of flexibility the application should have with respect to different contexts and how it should react to changes. Finally, as an illustration we have presented an application that uses the framework for supporting a trust-based selection of services dynamically.

References

[1] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - wsdl-s. http://www.w3.org/Submission/WSDL-S/, November 2005. W3C Member Submission.

[2] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, pages 263–277, June 2007.

[3] William J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289–350, 1985.

[4] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[5] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, 16(2):97 – 166, 2001.

[6] John Domingue, Liliana Cabral, Stefania Galizia, Vlad Tanasescu, Alessio Gugliotta, Barry Norton, and Carlos Pedrinaci. IRS-III: A broker-based approach to semantic Web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(2):109–132, 2008.

[7] Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema. http://www.w3.org/TR/sawsdl/, January 2007. W3C Candidate Recommendation 26 January 2007.

[8] Dieter Fensel, Mick Kerrigan, and Michal Zaremba, editors. *Implementing Semantic Web Services: The SESA Framework*. Springer, 2008.

[9] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2007.

[10] S. Galizia. WSTO: A Classification-Based Ontology for Managing Trust in Semantic Web Services. In *3th International Semantic Web Conference (ESWC 2006)*, Budva, Montenegro, June 2006.

[11] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[12] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[13] Markus Keidl and Alfons Kemper. Towards context-aware adaptable web services. In *WWW '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 55–65, New York, NY, USA, 2004. ACM.

[14] Panu Korpipaa, Jani Mantyjarvi, Juha Kela, Heikki Keranen, and Esko-Juhani Malm. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, 02(3):42–51, 2003.

[15] Reto Krummenacher, Holger Lausen, and Thomas Strang. Analyzing the Modeling of Context with Ontologies. In *International Workshop on Context-Awareness for Self-Managing Systems*, May 2007.

[16] Zakaria Maamar, Ghazi AlKhatib, Soraya Kouadri Mostefaoui, Mohammed B. Lahkim, and Wathiq Mansoor. Context-based Personalization of Web Services Composition and Provisioning. In *30th EUROMICRO Conference (EUROMICRO'04)*, pages 396–403, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[17] Zakaria Maamar, Djamal Benslimane, and Nanjangud C. Narendra. What can context do for web services? *Communications of the ACM*, 49(12):98–103, 2006.

[18] D. Martin, M. Burstein, Hobbs J., O. Lassila, D. McDermott, S. McIlraith, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2004.

[19] Enrico Motta. *Reusable Components for Knowledge Modelling. Case Studies in Parametric Design Problem Solving*, volume 53 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 1999.

[20] Enrico Motta and Wenjin Lu. A Library of Components for Classification Problem Solving. Ibrow (ist-1999-19005) deliverable, The Open University, 2001.

[21] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1):87–127, 1982.

[22] Barry Norton, Carlos Pedrinaci, John Domingue, and Michal Zaremba. Semantic Execution Environments for Semantics-Enabled SOA. *it - Methods and Applications of Informatics and Information Technology*, Special Issue in Service-Oriented Architectures:118–121, 2008.

[23] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[24] Carlos Pedrinaci, Christian Brelage, Tammo van Lessen, John Domingue, Dimka Karastoyanova, and Frank Leymann. Semantic business process management: Scaling up the management of business processes. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC) 2008*, Santa Clara, CA, USA, August 2008. IEEE Computer Society.

[25] Carlos Pedrinaci, John Domingue, and Ana Karla Alves de Medeiros. A Core Ontology for Business Process Analysis. In *5th European Semantic Web Conference*, 2008.

[26] A.Th. Schreiber and B.J. Wielinga. Configuration design problem solving. *IEEE Expert*, 12(2):49–56, April 1997.

[27] Mark Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann, San Francisco, CA, USA, 1995.

[28] Thomas Strang and Claudia Linnhoff-Popien. A Context Modeling Survey. In *First International Workshop on Advanced Context Modelling*, Nottingham, UK, 09 2004.

[29] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. CoOL: A Context Ontology Language to Enable Contextual Interoperability. *Distributed Applications and Interoperable Systems*, pages 236–247, 2003.

[30] Annette ten Teije, Frank van Harmelen, and Bob J. Wielinga. Configuration of Web Services as Parametric Design. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *EKAW*, volume 3257 of *LNCS*, pages 321–336, Whittlebury Hall, UK, October 2004. Springer.

[31] B. J. Wielinga, J.K. Akkermans, and G. Schreiber. A competence theory approach to problem solving method construction. *International Journal of Human-Computer Studies*, 49:315–338, 1998.

[32] Terry Winograd. Architectures for context. *Human-Computer Interaction*, 16(2):401 – 419, 2001.