Università degli Studi di Milano-Bicocca

SCUOLA DI DOTTORATO DI RICERCA IN SCIENZE

Dottorato di Ricerca in Informatica

XXII ciclo

Coordinatore Prof.ssa Stefania Bandini

WEB SERVICE CONTRACTS: SPECIFICATION, SELECTION AND COMPOSITION.

MARCO COMERIO

Ph.D. Dissertation

Supervisor:	Prof. Flavio De Paoli
Tutor:	Prof. Roberto Bisiani

A.A. 2008/2009

Abstract

Web services promise universal interoperability and integration of services developed by independent providers to execute business processes by discovering and composing services distributed over the Internet. This means that a key factor to build complex and valuable processes among cooperating organizations relies on the efficiency of discovering appropriate Web services and composing them. The increasing availability of Web services that offer similar functionalities requires mechanisms to go beyond the pure functional discovery and composition of Web services.

A promising solution towards the automatic enactment of valuable processes consists in enhancing Web service discovery and composition with the evaluation of semantic contracts that define non-functional properties (NFPs) and applicability conditions associated with a Web service. Nevertheless, currently there is a lack of tools and algorithms that fully support this solution due to several open issues. First, existing languages do not show the expressivity necessary for Web service contract specifications. Second, there is a lack of standard languages that determines heterogeneity in Web service contract specifications raising interoperability issues. Third, pure semantic approaches to enhance Web service discovery allows for detailed descriptions but present performance problems due to current limitations of semantic tools when dealing with reasoning tasks. Fourth, Web service contract compatibility evaluation is not supported by existing composition tools when combining different services from different providers.

This dissertation addresses these open issues and proposes solutions in terms of models, algorithms and tools.

Acknowledgements

I want to thank my supervisor Prof. Flavio De Paoli for giving me the opportunity to work on this thesis, and for his precious advice and ideas.

A special thank goes to Prof. Carlo Batini for giving me the opportunity to work in an inspiring research group.

Thank to the SEQUOIAS colleagues, and friends, Matteo Palmonari, Andrea Maurino and Luca Panziera for the joint works on the PCM and on the PoliMaR tool. Thank to the colleagues, and friends, of the SITI2 Lab Simone Grega, Gianluigi Viscusi, Giuseppe Frisoni and Pei Li that make easier to work in complex projects with strict time constraints.

Thank to Prof. Shahram Dustdar who gave me the chance to be a visiting student in the Distributed Systems Group of the Vienna University of Technology (TU Wien) where I found precious insights for my dissertation. A special thank goes to Hong-Linh Truong for the joint work on the SeCO$_2$ and for his advise during my visiting in TU Wien. Thank to all the researchers, and friends, of the VitaLab: Emmanuel, Fei, Hung, Javier, Ernst, Martin V., Martin T., Lukasz and all the others.

And I want to thank all the others I probably forget in this list...

Contents

List of Figures

Chapter 1

Introduction

Service-Oriented Computing (SOC) is a computing paradigm that proposes services as basic constructs for the development of rapid, low-cost and easy-to-compose distributed applications in heterogeneous environments [1]. The visionary promise of SOC is a service ecosystem where application components are assembled with little effort into a loosely-coupled network of services to create flexible and dynamic business processes and agile applications that might span organizations and computing platforms.

In a service oriented domain, the properties that completely specify a service can be classified as (1) functional, (2) behavioral and (3) non-functional. *Functional properties* (FPs) represent the description of the service tasks in terms of operation signatures (operation names and input/output schema) or more sophisticated descriptions (e.g., finite state automata specifications [2]). *Behavioral properties* describe how the functionality of the service can be achieved in terms of interactions with the service (i.e., choreography) and as well in terms of functionality required from the other services (i.e., orchestration). *Non-functional properties* (NFPs) represent the description of the service characteristics (e.g., availability, performance, price) that are not directly related to the functionality provided.

The Web service (WS) technology enables the implementation of the SOC paradigm and provides for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols [3]. The Web Service Description Language (WSDL) is the standard XML-based language used for describing Web services in terms of offered functionalities. The Simple Object Access Protocol (SOAP) is the protocol used for exchanging structured information in large distributed environments. The Universal Description, Discovery and Integration Protocol (UDDI) provides a mechanism for service providers to register their Web services and for service consumers to find them. The Business Process Modeling Language for Web Services (BPEL4WS) provides a language for the formal specification of business processes and behavioral properties.

Web services promise universal interoperability and integration of services developed by independent providers to execute business processes by discovering and composing services distributed over the Internet. Web service *discovery* is the activity of locating a machine-processable description of a Web service that meets certain functional criteria [4]. Web service *composition* consists in combining the functionality of several discovered Web services in order to define composite services [5]. This means that the key to build complex and valuable business processes among cooperating organizations relies on the efficiency of discovering appropriate Web services and composing them.

Nevertheless, there is a growing consensus that pure functional discovery and composition of Web services are inadequate to develop valuable processes due to the high degree of heterogeneity, autonomy, and distribution of the Internet [6]. The increasing availability of Web services that offer similar functionalities with different NFPs increases the need for more sophisticated Web service discovery and composition to match user requests [7]. In fact, even if a Web service matches the requested functionalities, it could be unacceptable for a specific user in terms of NFPs (e.g., if availability is not sufficient, performance is too poor, price is too expensive).

A possible solution to improve the development of valuable business processes consists in enhancing Web service discovery and composition with the evaluation of NFPs. The Web service discovery activity can be enriched with a Web service *selection* phase in which discovered services are evaluated in order to identify the one(s) that better fulfills a set of NFPs requested by the actual user [8]. The Web service composition activity can be extended with a new phase dealing with the evaluation of compatibility among NFPs offered by the discovered services [9–11].

Even if these NFP-based approaches appear to be proper solutions to improve the definition of valuable business processes, currently they are not applicable due to the lack of standards for NFP descriptions. Non-functional properties are often specified in the understandings between a service consumer and a service provider that can be established using different approaches (e.g., policies [12], licenses [13], service level agreements [14]). Even if some philosophical differences exist among these approaches, the common term *service contract*, which specifies conditions that a service consumer and a service provider agree, is used to identify them [15–19]. Besides a functional and behavioral description of the service usage, a service contract includes the specification of different types of NFP such as *Quality of Service (QoS)*, *Business*, *Service Context* and *License* terms of a service. *QoS* terms represent technical issues of the service (e.g., security and performance). *Business* terms describe financial terms and conditions (e.g., price, insurance and compensation agreements). *Service Context* terms define technical aspects (e.g., compliancy to available devices and connections) as well as profile aspects (e.g., service coverage) of the context associated with the service. Finally, *License* terms (e.g., compliancy to legal statements and regulations) state responsibilities among involved parties and conditions on service usage.

The heterogeneity of service contract specifications prevents their direct usage in automatic Web service discovery and composition since they are not comparable. A possible solution to this problem consists in using Semantic Web techniques to give unique interpretation and processable format to information extracted from existing heterogeneous service contracts and to represent them in a comparable format. The resulting semantic Web service contracts allow for efficient comparisons based on reasoning tasks to determine the relationships between contractual terms [20].

This dissertation proposes (i) a meta-model allowing sophisticated descriptions of semantic Web service contracts, (ii) techniques to extract information from service contracts defined in different languages and (iii) innovative approaches to semantic Web service contract selection and composition enabling the realization of valuable business processes. A real case-study in the logistic operator domain is used as a fully-featured test for the evaluation of the proposed solutions.

1.1 Motivation and Background

The enhancement of Web service discovery and composition with the evaluation of semantic contracts is recognized as a promising solution for the automatic definition of valuable processes composing services distributed over the Internet [9, 21]. Despite of that, there is the lack of tools and algorithms providing a complete solution due to several open issues. In this dissertation, four problems are addressed.

The Web service contract specification problem: existing languages for the service contract specification lack in the required expressivity to fully describe the complex nature of non-functional properties [12–14].

The Web service contract heterogeneity problem: the lack of standard languages determines heterogeneity in service contract specifications. This means that, even if these specifications are available, often they cannot be used to support Web service discovery and composition activities since they are not comparable.

The Web service contract selection problem: current approaches to improve Web service discovery with a selection phase based on the evaluation of NFPs specified in Web service contracts present some weaknesses. Non-semantic approaches (e.g., [22, 23]) are characterized by low precision due to only syntactic NFP descriptions. Semantic approaches (e.g., [21, 24]) present performance problems due to current limitations of semantic tools when dealing with reasoning tasks on NFP values.

The Web service contract composition problem: existing composition tools do not support the evaluation of Web service contract compatibility when combining different services from different providers.

1.1.1 The Web service contract specification problem

Besides a functional and behavioral description of the Web service usage, a service contract includes the specification of different types of NFPs concerning a variety of service aspects, such as *Quality of Service (QoS)*, *Business*, *Service Context* and *License* terms [25].

Current approaches for service contract specification (e.g., policies [12], licenses [13], service level agreements [14]) lack in providing support for expressive NFP descriptions. The description of NFPs is a complex task due to the nature of these properties. Several aspects must be considered:

- **synonym and homonym**: similar properties may have different names (e.g., in different languages or domains) or the same name may refer to different properties (e.g., in different domains a property may have different implications).

- **quantitative and qualitative values**: NFPs can be expressed with numeric values defined in different units (e.g., price in Euro or in USD), or they can be purely qualitative (e.g., the usability is *good*, trust is *high*, software is *open source*).

- **dynamic property values**: sometimes the value of a NFP is not fixed but needs to be calculated at run-time when the service context provides additional information (e.g., the bandwidth a service is provided with).

- **technological and business interdependencies**: clusters of NFPs need to be considered because of technological or business interdependencies among these properties (e.g., a higher price for the service that guarantees a certain bandwidth). In general terms, a service can provide the same functionality with different NFP levels. Several real world services provide evidence of this consideration. For example, the Italian railway company Trenitalia[1] offers different fares (e.g, Amica, Family, Same-Day-Return offers) for booking ticket services. Each fare is characterized by a set of NFPs such as *discount* and *advantages* and by applicability conditions. As an example, the Amica offer tickets with a discounted price when booking by midnight of the day before the departure.

- **requestor perspective**: service requestor must be supported in the definition of his/her NFP requirements. Advanced mathematical operators (e.g., '$\leq$', '$\geq$', range) must be allowed for requested NFP specification as well as a *relevance* value stating if an NFP is mandatory or optional.

Currently, many available models specify non-functional properties by means of <*attribute*,*value*> clauses (e.g., [26, 27]), where *attribute* identifies the involved NFP and *value* specifies the value of the property offered by the associated service. Shortcomings of such approaches are:

1. Attributes are usually labels that are defined by textual descriptions in natural language; there is not a formal definition of measurement methods and units. Moreover, qualitative dimensions are not standardized. Therefore, there is an high probability of semantic misunderstandings.

[1]http://www.ferroviedellostato.it/homepage_en.html

2. Offered properties are represented by single static values. Range-based offers cannot be easily expressed.

3. The representation of NFPs whose values are computed dynamically are not directly supported. In fact, neither the possible values of the properties, nor the methods used to assign them can be specified by means of attribute-value clauses.

4. Interdependencies among quality dimensions cannot be expressed since every dimension is considered in isolation.

5. Properties cannot be easily clustered to form a joint offer.

6. The capability to express NFP requested by users is limited. Advanced mathematical operators and relevance values are not usually supported.

In order to cover the above mentioned aspects and to solve the limitations of attribute-value models, a semantic meta-model that provides a sound and robust base to formally describe NFPs must be used.

Current standards for semantic descriptions of services (e.g., WSMO [27] and OWL-S [26]) only marginally cover the specification of NFPs. Several papers [4,28,29] propose solution to overcome WSMO and OWL-S limitations by specifying NFP through axioms. Other papers (e.g., [30–32]) propose semantic models for NFP descriptions that are not related to a particular standard. Even if these proposals succeed in covering some of the above mentioned aspects to be considered in NFP descriptions, none of them provides a solution able to cover all the identified aspects. Details about these works will be provided in Section 3.3.

1.1.2 The Web service contract heterogeneity problem

Currently, service contracts are established using different approaches (e.g., policies, licenses, service level agreements) and with different specification language, such as the Web Service Level Agreement (WSLA) language [14], Web Service Policy (WS-Policy) language [12], the Open Digital Rights Language for Services (ODRL-S) [13] and the Web Service Offerings Language (WSOL) [33].

Even though these languages allows for the specification of similar information, there exists no shared reference ontology/thesaurus for describing service contracts. This means that service consumers and providers can specify their service contracts as they wish, raising interoperability issues due to the impossibility to perform automatic matching when multiple services referring to heterogeneous service contracts are to be processed (e.g., in a composition).

The above mentioned languages for the specification of service contract can be classified in three categories:

- *Type A*: languages (e.g., ODRL-S, WS-Policy) allowing for the specification of a limited set of predefined properties, described in a profile model.
- *Type B*: languages (e.g., WSLA) allowing for the specification of user-defined properties. The user can introduce new properties without any reference to external definitions (e.g., by an ontology or a thesaurus).
- *Type C*: languages (e.g., WSOL) allowing for the definition of semantic service contracts through the specification of properties defined in ontologies, which can be standard or customized.

Each type of language presents weaknesses. Service contracts in Type A languages can be automatically processed/evaluated but the limited set of properties defined in fixed profile models prevent from covering all possible situations. Service contracts written in languages of Type B and C cannot be automatically processed/evaluated when the properties included by the users are unknown by the processing system. This problem is unsolvable in the case of languages of Type B, while it is somehow addressed for languages of Type C due to the use of ontology definitions; but even in this case, if there is not a meta-model behind the ontology, the processing becomes unreliable.

These observations bring to the conclusion that a Type C language based on an expressive meta-model is needed to fully describe NFPs in service contracts and allow interoperability. Moreover, techniques to perform the mapping of existing service contracts defined in different languages to the new defined language are needed in order to reuse available specifications.

Currently, no comprehensive solutions for *the Web service contract heterogeneity problem* are available in the literature. Ontology alignment tools [34–36], defining mappings between concepts in different ontologies, cannot be used to fully automate the mapping between heterogeneous service contracts because *Type A* and *Type B* specifications are not supposed to be based on ontologies. New proposals available in the literature (e.g., the VieSLAF [37]), represent a partial solutions to the problem since only mappings between particular specifications (e.g., service level agreement) are supported.

1.1.3 The Web service contract selection problem

Web Service discovery is a process that consists in the identification of the services that fulfill a set of requirements given by a user. Since more than one service is likely to fulfill the functional requirements, some ranking mechanisms are needed in order to provide support for the automatic or semi-automatic selection of a restricted number of services (usually one) among the discovered ones. The selection consists in identifying the Web services that offer contracts that better fulfill the NFP requirements given by the user. The evaluation consists in the computing satisfaction degrees between the set of requested NFPs and the set of NFPs offered by each discovered Web service and specified in its service contract.

Currently, Web service contract selection is executed using non-semantic or semantic approaches.

Non-semantic approaches (e.g., [22, 23]) are characterized by high time efficiency but low precision due to the management of only syntactic service contract descriptions. The evaluation of satisfaction degrees between qualitative NFPs is reduced to the syntactical comparison among values, raising semantic misunderstandings and inefficient selections.

Semantic approaches (e.g., [21, 24]) utilize automated reasoning techniques on semantic service contract descriptions. These techniques are particularly suitable to mediate between different terminologies and data models considering the semantics of the terms used in the descriptions as defined by means of logical axioms and rules (e.g., at class-level, by making explicit that, in a given domain, the property *BasePrice* is equivalent to the property *ServicePrice*, or, at instance-level, by making explicit that a *fire insurance* is part of a *blanket insurance*). Therefore, reasoning techniques can be used for the evaluation of satisfaction degrees between qualitative NFPs in order to exploit semantic relations between NFP values. However, the NFP evaluation based on logical reasoning conflicts with the need to support selection algorithms with more practical evaluation techniques since many reasoners show poor effectiveness when dealing with non trivial numeric functions (e.g., weighted sums) which are needed to manage more properties at the same time.

Moreover, the most relevant semantic or non-semantic approaches available in the literature [21–24, 38–40] present limitations in some of the following features:

- **expressivity**: the possibility to perform the selection process on service contract including expressive descriptions of requested and offered NFPs addressing qualitative properties by mean of logical expressions on ontology values and quantitative properties by mean of expressions including ranges and inequalities;
- **generality**: the possibility to create semantic-based mediation between NFP descriptions based on multiple ontologies;
- **extensibility**: the possibility to define parametric property evaluation by customizing evaluation functions;
- **flexibility**: the possibility to perform evaluation in case of incomplete specifications (i.e., unspecified properties and values in NFP requests and offers).

As a consequence, pure semantic or non-semantic approaches appear to be inadequate to solve *the Web service contract selection problem*. The possibility to combine logic-based and algorithmic techniques to provide for an effective and flexible approach to service contract selection offering good levels of expressivity, generality, extensibility and flexibility must be investigated.

1.1.4 The Web service contract composition problem

In the current service composition landscape, it is not so difficult for users to compose different services based on published service interfaces. For example, existing platforms like The Process Factory[2] and Boomi[3] provide users with different connectors for users to compose their services from various providers. These platforms allow users to combine different services, potentially characterized by different service contracts. However, in order to define legal processes, there is the need to ensure that the service compositions do not include conflicting service contracts. This assurance cannot be given by a single provider and currently is not available in existing composition tools.

[2]http://www.theprocessfactory.com
[3]http://www.boomi.com

Incompatibilities among *QoS*, *Business*, *Service Context* and *License* terms specified in contracts of services involved in the composition can determine the impossibility to execute the composite service. As an example, services offering incompatible values in *Data Ownership* (e.g., *copyrighted data* and *free data distribution*) or in *Service Coverage* (e.g., *Europe only* and *US only*) can cause inefficient composite service execution since their data are protected by different rules and their coverage is limited to different world regions.

Besides the evaluation of compatibility among service contracts, a unified contract for the composite service must be defined composing the contracts offered by the services involved in the composition. This unified contract specifies recommended property values, evaluated on the basis of the property values included in the offered contracts. As an example, supposing that compatible values for *Data Ownership* (e.g., *copyrighted data* and *personal data usage*) and *Service Coverage* (e.g., *Europe only* and *Italy only*) are specified, a proper value for these properties must be evaluated and included into the composite service contract.

Service contract compatibility and composition are complex activities since they must be evaluated according to the structure of the service composition. This is related to not only the *control flow* (i.e., the sequence in which the services are invoked) but also the *data flow* (i.e., the exchange of data between services) of the service composition.

While certain works [9–11] address QoS-based compatibility for control flows, currently there is not a good understanding of how to check contract compatibility and composition for data, the input/output of services, whose contract terms are not always the same to that of the services.

The consideration of both control and data flows is essential to perform an efficient Web service contract compatibility evaluation and composition. Let consider the scenario in which a user wants to create a *Travel Agency Service* (TA) by composing a *Flight Booking Service* (FB), an *Hotel Reservation Service* (HR) and a *Payment Service* (PS). The services follow a sequential execution and data are exchanged between FB and PS and between HR and PS. To evaluate the compatibility of the *availability time range* term (i.e., the time range in which the service is available) in the FB contract, the same term in HR contract must be considered since FB and HR are executed one after the other. Viceversa, to evaluate the compatibility of the *data ownership* term (i.e., a license term stating how the data produced by the service are protected) in the

FB contract, the same term in PS contract must be considered since FB data are managed by PS.

Past research has not focused on tools and algorithms dealing with contract compatibility evaluation when combining different services from different providers. Typically, they deal with only contract negotiation between consumer and service in a point-to-point manner. Examples are service customization approaches [41–43] that ensures for the compliance between the customization capability (i.e., the possibility to customize functional, non-functional and behavioral service aspects) defined by service providers and the requirements specified by the user, while the compliance between multiple contracts defined on possibly different set of properties is not tackled.

As a consequence, the definition of tools and algorithms dealing with service contract compatibility and composition evaluation considering data and control flows of the service composition appear to be an innovative research challenge.

1.2 Main contributions

This dissertation provides for a deep and broad examination of the above cited problems. Among the arguments discussed, the most significative contributions are:

- the definition of the Policy-Centered Metamodel (PCM), a meta-model for the description of expressive NFPs to be included in service contracts. The PCM allows for descriptions covering all the aspects identified in *the Web service contract specification problem*. A full set of constraint operators and *relevance* attributes are used for sophisticated descriptions of quantitative and qualitative NFP offers and requests. Moreover, NFP clustering is supported in order to capture technological and business interdependencies among properties.

  The work about the PCM has been partially and in different forms presented in [44, 45].

- the definition of techniques to perform the mapping from available service contracts defined in different languages (WSLA, WS-Policy, ODRL-S, WSOL) to PCM-based Web service contract specifications. These techniques represent a possible solution to *the Web service contract heterogeneity problem*.

The work about these techniques has been partially presented in [46].

- the design and implementation of the Policy Matchmaker and Ranker (PoliMaR), a tool allowing for an effective and flexible Web service contract selection that can be used to improve the traditional Web service discovery process. The approach used by the PoliMaR tool combines logic-based and algorithmic techniques and offers high levels of expressivity, generality, extensibility and flexibility. The PoliMaR tool represents a possible solution to *the Web service contract selection problem*.

  Preliminary works about service contract selection have been proposed in [47–51]. The work about the PoliMaR tool has been partially presented in [52].

- the design and implementation of the Service Contract Compatibility (SeCO$_2$) Evaluator, a tool for contract compatibility evaluation and composition that can be used to improve the traditional functional-based Web service composition process. The SeCO$_2$ Evaluator evaluates service contract compatibility and composition considering data and control flows of the service composition and represents a possible solution to *the Web service contract composition problem*.

  The work about the SeCO$_2$ Evaluator has been partially presented in [46].

1.3 Dissertation outline

The dissertation is composed of the following chapters. Chapter 2 describes the Logistic Operator Domain that will be used in the rest of the dissertation as a case study. Chapters 3-6 address the four problems introduced above by discussing the main issues, describing the solutions, and presenting the related works. Chapter 3 provides details about the Policy Centered Meta-model. Chapter 4 describes techniques for modeling PCM-based service contract specifications and for mapping available service contracts defined in different languages to PCM-based specifications. Chapter 5 is dedicated to the design, implementation and evaluation of the PoliMaR tool. Chapter 6 focuses on design, implementation and evaluation of the $SeCO_2$ Evaluator. Finally, Chapter 7 provides concluding remarks and gives an outlook to the future works.

Chapter 2

A Case Study

A structured knowledge-intensive description of a real domain is useful to understand the complexity of the problems to be addressed and to provide a fully featured test for the proposed solutions. The logistic operator domain shows the required features since it is characterized by: (i) well-known terminology; (ii) well defined processes; (iii) articulated normative; (iv) different points of views between service providers and consumers.

The case study has been defined within the Networked Peers for Business (NeP4B) project [1] through (i) the analysis of several real-world logistic operator prices and service offer lists, (ii) several phone and face-to-face interviews with logistic operators and client companies and (iii) the analysis of the normative that regulates the logistic operator activities. Moreover, this description has inspired a new scenario for the Semantic Web Service Challenge 2009[2] described in [53]. Therefore, the case study can be considered representative of real-world logistic processes.

This chapter proceeds as follows. Section 2.1 describes the most interesting characteristics of the logistic operator domain. Section 2.2 defines a scenario that will be used as a running example in the rest of the dissertation.

2.1 The Logistic Operator Domain

In the logistic domain there are two main actors (namely, *logistic operator* and *client company*) each one with his features and objectives. The primary objective of a logistic operator is to provide logistic support to third parties. Client

[1] http://www.dbgroup.unimo.it/nep4b/en/index.htm
[2] http://sws-challenge.org/wiki/index.php/Scenario:_Logistics_Management

companies, instead, request services to logistic operators for their transportation.

For example, the logistic support offered by logistic operators might concern freight transport and warehousing which are complex and structured services. In freight transport service, the logistic operator makes a delivery of a freight, in an agreed location, within some time constraints. In warehousing service, the logistic operator stores a certain amount of goods for some time in one of its warehouse. A logistic operator can provide only freight transportation service or only warehousing service or both the services, even in combination.

Freight transport and warehousing regard different categories of goods such as (i) ordinary; (ii) perishable; (iii) dangerous and (iv) cumbersome. Each category must be treated in different ways and needs a specific means of transportation (i.e., classes of trucks). Mainly, the categories of goods are: (i) ordinary; (ii) perishable; (iii) dangerous and (iv) cumbersome. Ordinary goods do not need a specific way or particular attention to be treated. They can be transported by ordinary trucks, with no particular features. An ordinary load can be transported aggregated with other kinds of ordinary goods (groupage). Perishable goods, instead, need a particular temperature that must be maintained for all the transport duration. Perishable goods can be fresh, frozen or deep-frozen and need to preserve different temperature during the transportation. This means that not every perishable goods can be transported together with other ones. Dangerous goods, need a particular planning of the freight and particular treatment. Cumbersome goods are particular loads characterized by big dimensions and weight. The transport of this kind of good is regularized by the local rules of the road (exceptional transport).

This means that, logistic operators can offer a logistic support service (freight transport or warehousing) associated with several service contracts according to the category of the goods to be transported/stored. Moreover, client companies can ask for logistic support services specifying additional features for the transportation/storage.

The problem is that different points of view are used by logistic operator and client company when describing offers and requests of a logistic operator service. In fact the logistic operator has its terminology for the description of the services that a client company may not know or understand.

Typically, a logistic operator offer is characterized by:

- *a fleet*: identifying the kinds of goods that a logistic operator can treat since specific normative states particular procedures and freight truck for every kind of goods. Other features such as carrying capacity of the trucks in the fleet may attest the ability of a logistic operator to fulfill a client company request;
- *a logistic platform*: identifying the kinds of goods that a logistic operator can store, the minimum and maximum load dimensions and the location of the warehouse;
- *covered geographic area*: representing where a logistic operator can pick-up or deliver a load;
- *time constraints*: specifying the minimum quantity of hours to plan the freight;
- *a base price*: specifying the cost for the service invocation;
- *accepted payment methods*: defining if pre-paid or post-paid payment methods (i.e., the freight bill is at sender's or addressee's expense) are accepted;
- *additional services*: specifying additional features such as freight insurance, COD (Cash On Delivery), express deliveries or deliveries with precise and mandatory delivery instant.

A client company may express its need in a whole different way. A typical request is characterized by:

- *a quantity of a particular kind of goods*: the load to be transported;
- *pick-up and delivery locations*: the address and the kind of location of the freight (warehouse, domicile, etc.);
- *time constraints*: a start and an end time identifying the interval between the pick-up and the delivery of the load;
- *a payer*: at who's expense is the freight bill (i.e., the addressee or the sender);

- *additional features*: preferences such as the possibility to aggregate the load with other loads, the freight insurance, the delivery time and particular truck features.

2.2 A Reference Scenario

In order to discuss and test the solutions defined in this dissertation, the process of purchase data analysis inside a supply chain management scenario is considered. This process, characterized by a set of interrelate tasks (e.g., data, payment and shipment validations), can be automatized through the selection and composition of services covering the tasks. A possible automatization of the purchase data analysis process consists in composing the following list of services that includes, among others, a freight transportation service:

- a `Request Service` (RS) issuing a purchase request;
- a `Purchase Processing Service` (PPS) managing the standard e-commerce process;
- a `Merchant Validation Service` (MVS) verifying and providing data about a shopping merchant;
- a `Payment Verification Service` (PS) validating data related to the payment (e.g., the credit card number);
- a `Freight Transportation Service` (FTS) offering transportation of every kind of goods;
- a `Purchase Validation Service` (PVS) analyzing data and validating the purchase.

As shown in Figure 2.1, the composite service can be defined using different composition structures allowing for different control and data flows between the selected services.

Starting from the description of the purchase data analysis process, an instance for each problem specified in Section 1.1 can be defined.

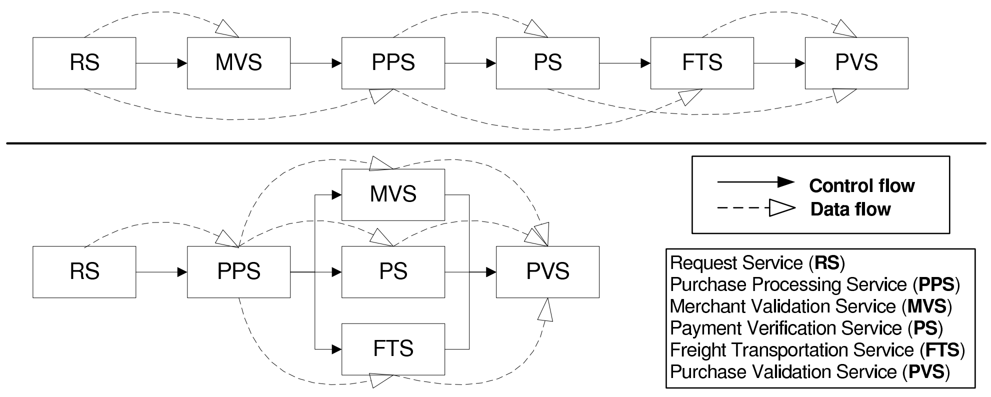

RS
MVS
PPS
PS
FTS
PVS
Control flow
Data flow
Request Service (RS)
Purchase Processing Service (PPS)
Merchant Validation Service (MVS)
Payment Verification Service (PS)
Freight Transportation Service (FTS)
Purchase Validation Service (PVS)

Figure 2.1: Composition structures for the *Purchase Data Analysis* service
.

2.2.1 Web service contract specification

Different Web service contracts can be associated with the same `Freight Transportation Service` (FTS). These contracts can differ in terms of non-functional properties (NFPs), such as:

- *base price*: the amount of money that must be paid for each freight independently from specific features of the order;
- *payment method*: how the user can perform the payment. Common payment methods are:
  - *carriage paid*: the seller pays the freight for the delivery;
  - *carriage forward*: the receiver pays the freight for the delivery.
- *payment deadline*: the maximum number of days that the user can wait to perform the payment after the service fulfilment;
- *insurance*: the type of prevention applied to the service;
- *hours to delivery*: the number of hours required for the service fulfilment.

The definition of expressive descriptions for these NFPs is necessary in order to allow (i) logistic operators to fully cover the complex specifications of offered FTS contracts and (ii) client companies to define articulated FTS contract requets (see Figure 2.2).

A logistic operator service provider can specify the following contract (namely, *premium policy*) associated with its FTS service:

"I offer a service that performs freight transportation in 24-48 hours with a base price equal to 100 Euros. I accept carriage paid payment within 45 days and I offer a blanket insurance on the transportation".

A client company might want to formulate quite rich requests to identify the best service according to their own stated criteria. An example of request (namely, *LORequest1*) written in natural language is the following:

"I am interested in a service to perform a freight transportation in one or two days with a price less than or equal to 120 Euros. Moreover, I would like to use a service allowing, at least, a 15-days postponed payment with carriage paid or carriage forward payment method. Finally, I prefer a service offering a fire insurance or any insurance type that includes it".

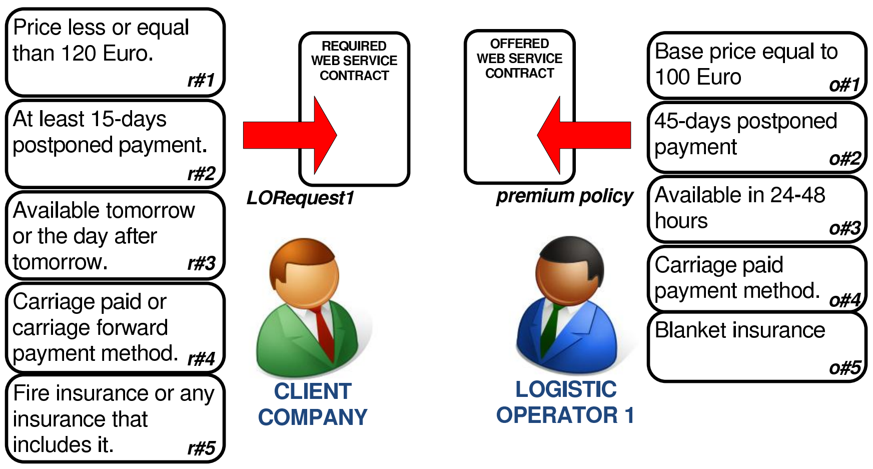

Price less or equal than 120 Euro. r#1
At least 15-days postponed payment. r#2
Available tomorrow or the day after tomorrow. r#3
Carriage paid or carriage forward payment method. r#4
Fire insurance or any insurance that includes it. r#5
REQUIRED WEB SERVICE CONTRACT
LORequest1
CLIENT COMPANY
OFFERED WEB SERVICE CONTRACT
premium policy
LOGISTIC OPERATOR 1
Base price equal to 100 Euro o#1
45-days postponed payment o#2
Available in 24-48 hours o#3
Carriage paid payment method. o#4
Blanket insurance o#5

Figure 2.2: An example of WS contract description problem

In this scenario different types of NFP have been included. NFPs may refer to either numerical values (e.g., 120) or world objects (e.g., *fire insurance*) and some values can be undefined in the requests (e.g., in a lower bound expression such as *price less than or equal to 120 Euros*) or even in the offered NFPs (e.g., in a range expression such as *24-48 hours*).

In particular, the NFPs specified above can be classified as qualitative (*payment method*, *insurance*) or quantitative (*payment deadline*, *base price*, *hours to delivery*).

Qualitative NFPs are expressed with values that can be defined as ontological instances. As an example, the terms *carriage paid* and *carriage forward* used to specified the *payment method* can be defined in an ontology where all the logistic operator concepts (and their relations) are specified. Qualitative NFPs are specified using different operators to add more semantic to their specifications. As an example, a user can specify a set of values and states that he/she is interested in services that offer at least one of the requested types (e.g., *carriage paid or carriage forward payment method*), exactly all the requested types, or any type that includes the requested ones (e.g., *a fire insurance or any insurance type that includes it*). The latter example underlines the necessity to consider that a qualitative NFP can assume a set of values possibly defined in different ontologies and possibly characterized by semantic relation among them (e.g., *fire insurance* is characterized by a *partOf* relation with *blanket insurance*).

Quantitative NFPs are expressed by numerical values. They can be specified as a single value (e.g., *base price equal to 100 Euro*) or as a range of values defining the possible values that the property can assume at runtime (e.g., *freight transportation in 24-48 hours*). Also for the specification of quantitative NFPs different constraint operators can be used. Along with the most common operators (e.g., $=,<,>,\leq,\geq$), also others (e.g., at least, at most) should be supported. Moreover, quantitative NFPs introduce the necessity to consider that each value is defined according to a unit of measure. Therefore, the possibility to have values with different units of measure must be considered.

Qualitative and quantitative NFP requests should be characterized by a relevance value in order to express preferences about what should be considered more important. These values allow to differentiate between expressions such as *"I am interested in"*, *"I would like to"* and *"I prefer"* that reveal different relevance for a NFP constraint.

As stated in 1.1.1, current standards for semantic descriptions of services (e.g., WSMO [27] and OWL-S [26]) only marginally cover the specification of NFPs. In order to cover the mentioned NFP characteristics, a semantic meta-model that provides a sound and robust base to formally describe NFPs must be defined.

2.2.2 Web service contract heterogeneity

Logistic operators and client companies can specify their offered/required FTS contracts using different approaches and languages. This raises interoperability issues due to the impossibility to perform automatic matching when multiple services referring to heterogeneous service contracts are to be processed.

Let consider the scenario in Figure 2.3. The Client Company and Logistic Operator1 specify *LORequest1* and *premium policy* using the same approach and language. Logistic Operator2 provides only a WSLA document (namely, *silver policy*) specifying, among others, the offered NFPs.

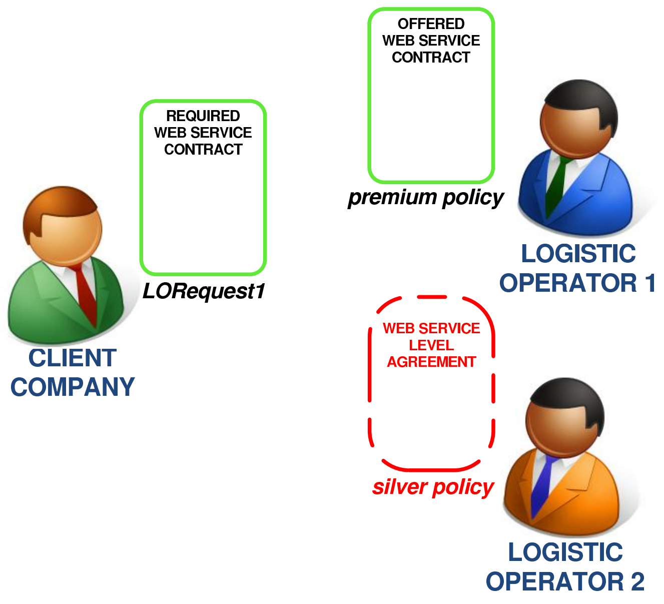


Figure 2.3: An example of WS contract heterogeneity problem

In order to make *LORequest1*, *premium policy* and *silver policy* comparable, proper techniques for extracting relevant information from the WSLA document and for mapping them in the same language used by the Client Company and Logistic Operator1 must be defined. Listing 2.1 shows an excerpt of the WSLA document offered by Logistic Operator2.

Listing 2.1: WSLA Specification

```
<?xml version="1.0" encoding="UTF-8"?>
...
<SLAParameter
          name="BasePrice"
          type="float"
          unit="euro">
```

```
</SLAParameter>
<SLAParameter name="PaymentDeadline"
        type="float"
        unit="days">
</SLAParameter>
<SLAParameter name="RequiredDeliveryTime"
        type="float"
        unit="hours">
</SLAParameter>
<SLAParameter name="PaymentMethod"
        type="list">
</SLAParameter>
<SLAParameter name="Insurance"
        type="list">
</SLAParameter>
...
<Expression>
  <And>
    <Expression>
      <Predicate xsi:type=equal>
        <SLAParameter>BasePrice</SLAParameter>
          <Value>150</Value>
      </Predicate>
    </Expression>
    <Expression>
      <Predicate xsi:type=equal>
        <SLAParameter>PaymentDeadline</SLAParameter>
          <Value>60</Value>
      </Predicate>
    </Expression>
    <Expression>
      <Predicate xsi:type=equal>
        <SLAParameter>RequiredDeliveryTime</SLAParameter>
          <Value>24</Value>
      </Predicate>
    </Expression>
    <Expression>
      <Or>
        <Expression>
          <Predicate xsi:type=equal>
            <SLAParameter>PaymentMethod</SLAParameter>
              <Value>"CarriagePaid"</Value>
          </Predicate>
        </Expression>
        <Expression>
          <Predicate xsi:type=equal>
            <SLAParameter>PaymentMethod</SLAParameter>
              <Value>"CarriageForward"</Value>
          </Predicate>
        </Expression>
      </Or>
    </Expression>
    <Expression>
      <Predicate xsi:type=equal>
        <SLAParameter>Insurance</SLAParameter>
          <Value>"fireInsurance"</Value>
      </Predicate>
    </Expression>
  </And>
</Expression>
```

2.2.3 Web service contract selection

Supposing to have solved *the Web service contract specification problem* and *the Web service contract heterogeneity problem*, there is the situation in which Client Company and the Logistic Operators specify comparable FTS contracts. The next problem consists in selecting the best FTS contract among *premium policy* and *silver policy* respect to the *LORequest1* specified by the Client Company.

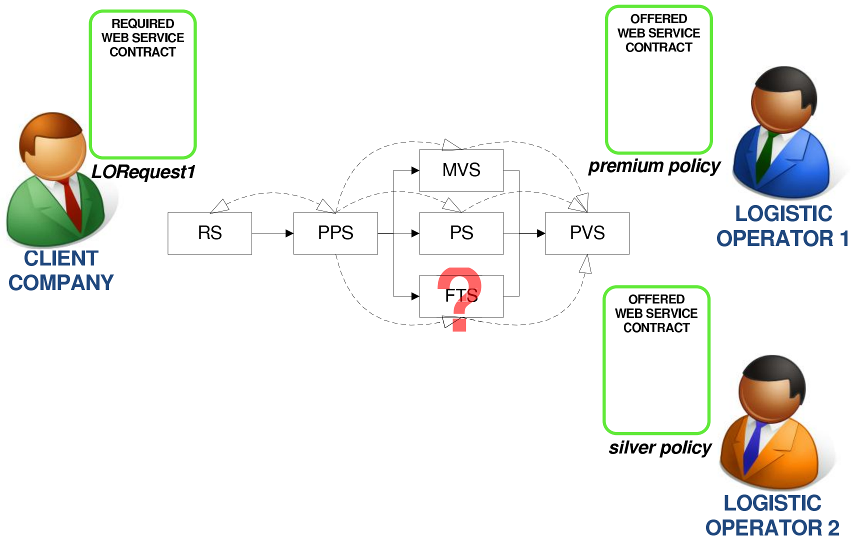


Figure 2.4: An example of WS contract selection problem

As defined in Section 1.1.3, the selection must be performed evaluating a degree of match between each offered FTS contract and the *LORequest1*. Since pure semantic or non-semantic approaches appear to be inadequate to solve *the service contract selection problem*, an approach based on logic and algorithmic techniques to provide for an effective, flexible and automatic service contract selection must be defined.

2.2.4 Web service contract composition

Assume that the selection activity to create the `Purchase Data Analysis` (`PDA`) service has produced the following Web service selection: (i) *Yahoo! Shopping Web Service*[3] has been selected as MVS; (ii) *XWebCheckOut Web Service*[4] as PPS; (iii) *Aivea Shipping Web Service*[5] as FTS; (iv) *ValidateCreditCard Web Service*[6] as PS and (v) *DOTS Lead Validation Web Service*[7] as PVS. The focus of this example is on service contracts. Thus, assume that these Web services match the functionalities required for the `PDA` service.

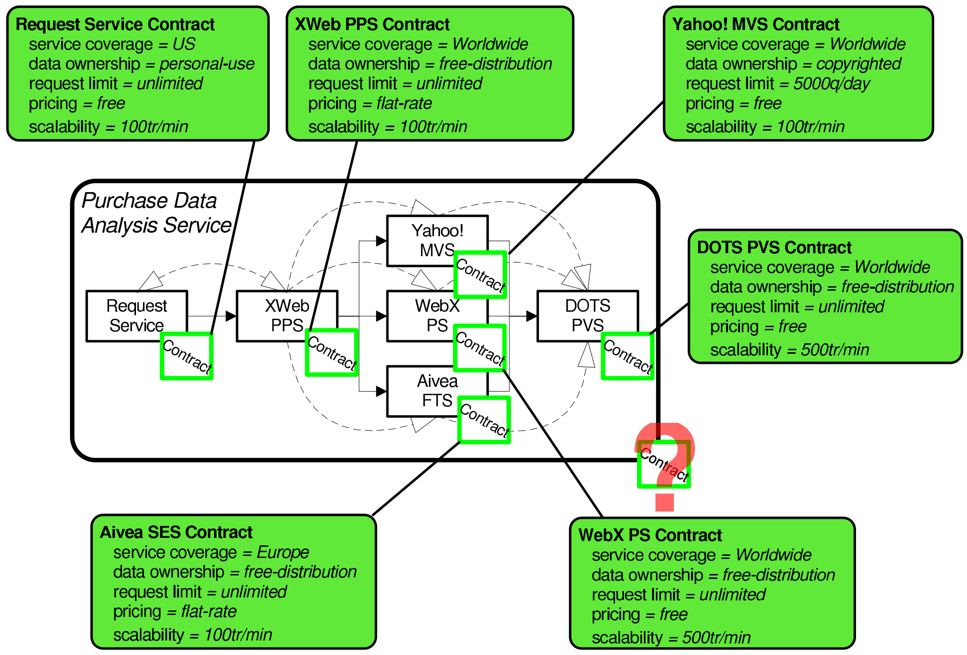

Request Service Contract
service coverage = US
data ownership = personal-use
request limit = unlimited
pricing = free
scalability = 100tr/min
XWeb PPS Contract
service coverage = Worldwide
data ownership = free-distribution
request limit = unlimited
pricing = flat-rate
scalability = 100tr/min
Yahoo! MVS Contract
service coverage = Worldwide
data ownership = copyrighted
request limit = 5000q/day
pricing = free
scalability = 100tr/min
Purchase Data Analysis Service
Request Service
XWeb PPS
Yahoo! MVS
WebX PS
Aivea FTS
DOTS PVS
Contract
DOTS PVS Contract
service coverage = Worldwide
data ownership = free-distribution
request limit = unlimited
pricing = free
scalability = 500tr/min
?
Aivea SES Contract
service coverage = Europe
data ownership = free-distribution
request limit = unlimited
pricing = flat-rate
scalability = 100tr/min
WebX PS Contract
service coverage = Worldwide
data ownership = free-distribution
request limit = unlimited
pricing = free
scalability = 500tr/min

Figure 2.5: An example of WS contract composition problem

As shown in Figure 2.5, the selected Web services are characterized by service contracts characterized by different values for the following properties:

- *Service Coverage*: a *Service Context* term defining world regions in which a service is available;
- *Data Ownership*: a *License* term stating how the data produced by the service are protected;

[3]http://developer.yahoo.com/shopping/V1/merchantSearch.html
[4]http://www.xwebservices.com/Web_Services/XWebCheckOut/
[5]http://www.aivea.com/shipping-web-service.htm
[6]http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=14
[7]http://www.serviceobjects.com/products/composite/lead-validation

- *Request Limit*: a *License* term defining the maximum number of requests that a user can submit to the service in a day;
- *Pricing*: a *Business* term describing possible pricing models offered by the service;
- *Scalability*: a *QoS* term specifying the maximum number of transactions accepted per minute

The first problem to be tackled consists in the evaluation of the compatibility between the above mentioned properties specified in service contracts associated with the selected RS, PPS, MVS, PS, FTS and PVS. After having solved the identified incompatibilities, *the service contract composition problem* proceeds with the definition of a contract for the composite service considering all the contracts of the services included into the composition. In order to evaluate how the structure of the composition can influence the service contract compatibility evaluation, these activities must be executed considering both the compositions represented in Figure 2.1.

The set of considered properties has been changed respect to the example used in the previous problems for two reasons: (i) to consider different NFP types (i.e., *QoS, Business, Context* and *License terms*) that can be included in Web service contracts. The NFP compatibility and composition must be evaluated considering data flow or control flow according to the NFP type; (ii) to underline that compatibility and evaluation can be performed on NFPs different from the ones used for the Web service contract selection. Incompatibilities can be identified also on NFPs not explicitly requested by the user but specified in selected service contracts.

Chapter 3

Web Service Contract Specification

Formally, a *contract* is a legally binding exchange of promises or agreement between parties that the law will enforce [54]. The contents of contracts may vary as the definition of contract is very broad in scope. In general, a contract includes an offer, the acceptance of the offer and an intention to create legal relations between parties.

In this dissertation, service contracts are considered. A *service contract* is a contract associated with a service describing how the usage of a service should be and including normative aspects that are agreed between the service consumer and the service provider who acted in compliance with the contract [17]. Currently, service contracts are established by several approaches across various application domains. The most common approaches consist in specifying *policies, service level agreements* and *licenses*.

Policies establish a relationship between involved parties, specifying obligations and authorizations. Obligations specify the set of activities that an object must or must not perform on target objects and authorizations specify the set of activities that an object is permitted or prohibited to perform on target objects [55].

A Service Level Agreement (SLA) is a bilateral statement signed between a service provider and a service consumer, over the agreed terms and conditions of the given service. A SLA describes the minimum performance criteria a provider promises to meet while delivering a service and typically sets out the remedial action and any penalties that take effect if performance falls be-low the promised standard. Thus, a SLA specifies the expected operational characteristics of a service in business oriented terms between a provider and a consumer, so that the characteristics can be measured, monitored, and managed [56, 57].

Licenses reflect the rights of the providers to control how the service is distributed and primarily focuses on the usage and provisioning terms of services [58]. Optionally, a service license can include SLA terms. Thus, a service license can be broader than a SLA, protecting the rights of service providers and service consumers.

This dissertation introduces the term *Web service contract* in order to decouple provider and consumer perspectives in contractual concerns of Web services. An offered *Web service contract* is a contract specified by a Web service provider defining functional/non-functional properties and applicability conditions associated with a Web service. A requested *Web service contract* is a contract specified by a Web service consumer defining requirements in terms of functional/non-functional properties associated with a Web service.

The separation of the two perspectives is required since dynamic and automatic selection and composition of Web services cannot be covered by the definition of predefined agreements on service usage between Web service providers and consumers. The definition of the final agreement is a process involving selection and composition tools to evaluate the compatibility between the contracts specifies by the two parties.

Since Web service contracts aim at specifying contractual terms in the Web service domain, these specifications must be *expressive*, *semantic*, *shared* and *machine-processable*. The aim of specifying contractual terms requires for high expressive and semantic descriptions in order to cover the complex nature of these terms and to avoid misunderstanding between specifications. The aim of using these contracts in the Web service domain requires for shared and machine-processable descriptions (such as WSDL documents) in order to be automatically used in Web service selection and composition activities.

As stated in Section 1.1.1, non-functional properties (NFPs) represent the most challenging contractual terms to be represented due to the nature of these properties. Focusing on that, the *Web service contract specification problem* can be solved with the definition of a semantic meta-model providing a sound and robust base to formally describe NFPs. The meta-model should be general and expressive enough to address the most significant issues in NFP descriptions and to support the matching of requested and offered properties. The meta-model should be independent from the language used for its specification.

This dissertation proposes the Policy-Centered Metamodel (PCM) as a solution for the *Web service contract specification problem*. The primary goal of the PCM is to support NFP descriptions according to the following characteristics:

- **Support for sophisticated descriptions**. The PCM refers to a full set of constraint operators (e.g., $\leq$, $\geq$ and interval) for the specification of NFP offers and requests. Moreover, *relevance* attributes allow users to specify the importance that requesters give to each requested NFP. The use of relevance attributes and constraint operators in NFP expressions (e.g., the cost of the service must be $\leq$ 3 Euro) enhance the expressivity of the descriptions to support non-boolean matching. In practice, as far as functional properties are concerned, a boolean match between requests and offered services is reasonable: a service that does not exactly fulfill the requested functionalities should be discarded. On the contrary, matching between offered and requested NFPs should be considered not crispy and degrees of satisfaction should be evaluated (i.e., a NFP request can be fully, partially or not satisfied).

- **Support for quantitative and qualitative NFP descriptions**. Since NFPs can be expressed with numerical values, the PCM supports the definition of units of measurement. Moreover, since NFP can also be purely qualitative, the meta-model supports the specification of ontological instances as NFP values.

- **Support for offer clustering**. The PCM allows clustering of NFPs in sets (i.e., *policies*). This is an important feature in order to capture business scenarios by aggregating interdependent properties. For example, a SMS service offered by a telephone company can be characterized by different offer clusterings: one offering a price of 0.10 Euro for SMS and a maximum of 100 SMS per day; another offering a price of 0.15 Euro and no restriction on the number of SMS per day. Moreover, such clusterings can be associated with *conditions* to state that their applicability depends on the requester's profile or context (e.g. a condition may grant a discount on shipment services to registered customers or for multiple shipment requests).

- **Support for static and dynamic properties description**. The PCM supports the specification of *static* NFPs representing properties of services characterized by fixed values that can be included in service contracts at publishing time. Moreover, the PCM supports also the specification of *dynamic* NFPs representing properties of services that are strictly related to the context of execution and cannot be stated at publishing time. A dynamic NFP is defined through list/range of possible values that the NFPs can assume at execution-time.

This chapter proceeds as follows. Section 3.1 provides a formal specification of the PCM. Examples of PCM-based Web service contract specifications are in Section 3.2. Finally, the comparison between the proposed solution and the related works is in Section 3.3.

3.1 The Policy-Centered Metamodel (PCM)

In this section the Policy-Centered Metamodel (PCM) is described by a BNF grammar[1] that provides a conceptual syntax whose semantics is defined by an upper-level ontology. The ontology has been formalized in the most popular semantic languages in the Web Service domain, namely OWL-DL and WSML. Each formalization provides a logical syntax that can be exploited to write actual descriptions of requested and offered NFPs.

3.1.1 PCM conceptual syntax

Figure 3.1 is a graphical representation of the ontology that illustrates the top-level concepts and relationships of the proposed meta-model. *Policy* is the main concept of the meta-model. Note that in the PCM this term assumes a different meaning respect to [55] since it must be considered as an NFP cluster characterized by the following elements: (i) *Id*: the URI that makes the policy available over the Internet; (ii) *name*: a given name that improves human readability, it can be used internally to identify a policy; (iii) *serviceReference*: the reference to the functional description of services the policy is attached to, e.g., the URI of a WSDL description; (iv) *policyCondition*: a logical element

[1]Conventions: [ ] optional, * one or more,

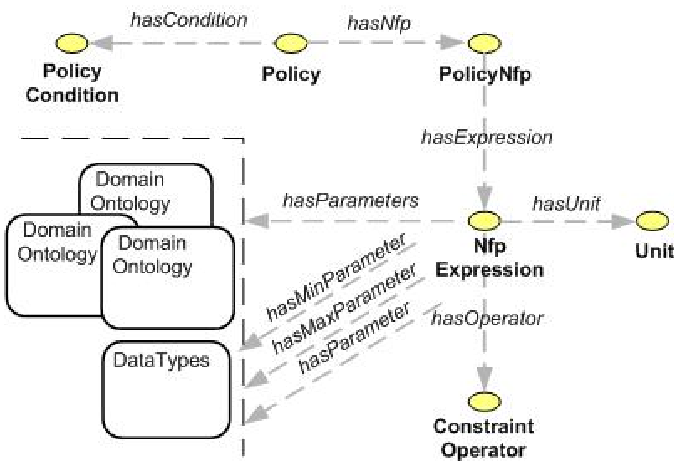


Figure 3.1: PCM top concepts

that defines the conditions for the validity of the policy; (v) *PolicyNfp*: one or more elements that specify the properties and their values.

The grammar to define instances of the *Policy* concept is specified in Listing 3.1:

Listing 3.1: Grammar for *Policy*

```
Policy := 'policy' Id
    PolicyHeader
    'PolicyNfp'*

Id := URI

PolicyHeader := 'name' String // label for human readability
    ['serviceReference' Id*]
    ['policyCondition' PolicyCondition]
    ['description' string] // textual description of conditions


PolicyCondition := Id {'definedBy' LogicalExpression}*

LogicalExpression := // axiom definition in the form of the
    selected semantic language
```

The *serviceReference* allows providers to link a policy to one or more Web service, independently from the chosen Web service description language. The aim of a policy condition is to define the conditions for client profile to select the policy (e.g., the *senior* policy is for clients older than 60). A policy condition is an instance of the *PolicyCondition* concept that can be further specified by *LogicalExpressions* that are axioms in the semantic language used to express the offer.

The core of the meta-model is the definition of the non-functional properties (*PolicyNfp*) by means of the following elements: (i) *Id*: the URI that makes the non-functional property available over the Internet; (ii) *NfpClass*: the ontology concepts of which this is an instance; (iii) *NfpExpression*: the expression that characterizes the properties and their values. The corresponding grammar is reported in Listing 3.2.

Listing 3.2: Grammar for *PolicyNfp*

```
PolicyNfp := 'nfp' Id
    NfpClass*
    NfpExpression

NfpClass := 'memberOf' Id // specifies the nfp class the
    property is member of

NfpExpression := SingleValueExpression | RangeExpression |
    SetExpression | CustomExpression

SingleValueExpression := BinaryOperator numericValue 'unit'
    unitValue

RangeExpression := TernaryOperator '(' numericValue,
    numericValue ')' 'unit' unitValue

SetExpression := SetOperator Id*

CustomExpression:= CustomOperator Id*

BinaryOperator := 'greaterEqual' | 'lessEqual' | 'equal' | '
    atLeast' | 'atMost'

TernaryOperator := 'interval'

SetOperator := 'all' | 'exist' | 'include'

CustomOperator := // user defined
```

Each *NfpExpression* is specified by a *ConstraintOperator* and by a set of attributes that depend on the constraint operator used. Consequently, starting from a classification of *ConstraintOperator*, a classification of *NfpExpression* and *PolicyNfp* has been defined. Figure 3.2 shows the logical relationships between the classifications. PCM distinguishes between qualitative and quantitative properties. The former are properties defined through qualitative expressions that refer to objects (their values are instances of given domain ontologies); the latter use quantitative expressions that assume numeric values, whose measurement units is specified by a 'unit' term.

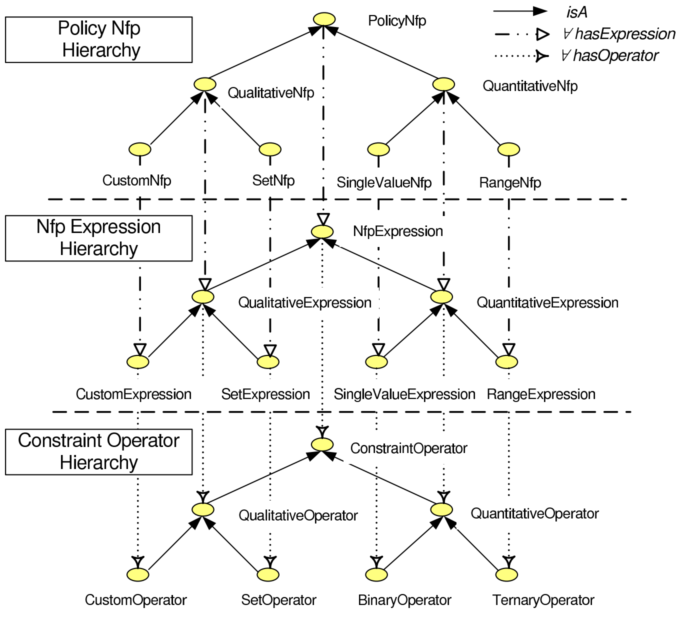


Figure 3.2: PCM hierarchies

The PCM introduces a set of logical operators of class *SetOperator*: (i) the two standard logical operators *all* and *exist* with their intuitive logical meanings, and (ii) the operator *include*. Intuitively, a *include*-based request (e.g., I need an insurance including *fire insurance*) asks for values that *logically* include the selected values (e.g., a *blanket insurance*); logical inclusion is looked up by exploring hierarchical properties of different nature (e.g., *part-of*, *topological inclusion*). The set of *CustomOperators* allows domain experts to introduce other operators to deal with object values. As an example, a request based on *semanticDistance* operator may ask for values that are semantically close to the specified one.

As for quantitative expressions, PCM defines a set of operators that supports the most common clauses for numeric values (e.g., inequalities and ranges). Beside the standard binary operator $=$ (*equal*), and ternary operator *interval* that fixes a minimum and a maximum value, new operators have been introduced to increase expressiveness of inequalities. These operators are: (i) $\geq\uparrow$ (*greaterEqual*) to specify a lower bound, so that the highest possible value is better; (ii) $\geq\downarrow$ (*atLeast*) to specify a lower bound, so that the lowest possible value is better; (iii) $\leq\downarrow$ (*lessEqual*) to specify an upper bound, so that the lowest possible value is better; (iv) $\leq\uparrow$ (*atMost*) to specify an upper bound, so that the highest possible value is better. Observe that binary operators are followed by one parameter and ternary operators by two parameters.

The counterpart of *PolicyNfp* is called *Request*. The aim of a *Request* is to state what values are acceptable for a certain property, and to express the relevance of the property for the user. As shown in Figure 3.3, *Request* is a subclass of *PolicyNfp* augmented with the definition of relevance values. *RequestedPolicy* is a subclasss of *Policy*, whose associated *PolicyNfp* are *Requests*. The BNF is in Listing 3.3.

Listing 3.3: Grammar for *RequestedPolicy*

```
RequestedPolicy := 'requestedPolicy' Id
    PolicyHeader
    Request*

Request := PolicyNfp 'relevance' RelevanceValue

ReferenceValue := Float // in range [0, 1]
```

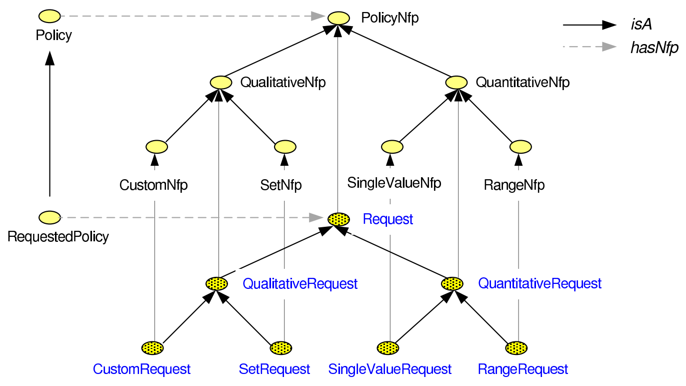


Figure 3.3: PCM concepts for requests

3.1.2 PCM ontological semantic

Although the meta-model has been designed to be independent from any specific language, the ontology defining its semantics has been formalized in the OWL-DL and WSML languages. OWL is in fact the language adopted by the semantic Web service model OWL-S and is recalled as mark up language in light semantic annotation languages such as SAWSDL [59]. WSML is a family of languages used for the Web Service Modeling Ontology WSMO. Besides the differences between WSML and OWL-DL, the two formalization of PCM are largely equivalent but for small details.

In this section only the definition of main concepts is given. The complete WSML formalization is reported in Appendix A and is available at the Website: http://www.siti.disco.unimib.it/?page_id=193/.

The first formalization of the PCM is in WSML because of the WSMO meta-model natively includes the distinction between service offer and service request. The latter is modeled by the concept of goal. Goal descriptions have the same schema of WS descriptions; that is, they are described mainly by capabilities (preconditions, effects, assumptions, postconditions), and annotations.

WSML-Flight is a language based on frame-logic and Logic Programming (LP); syntax of LP rules in WSML-Flight is a Object Oriented (or ontological) notational variant of normal logic programs (Prolog-like Horn Logic). WSML-Flight defines a conceptual syntax over the Frame Logic based core to provide a more user friendly description language. The syntax used for the meta-model formalization will be explained together with the axiom description.

The WSML formalization of the PCM top concepts is reported in Listing 3.4 where the literals *hasX* are used to give an intuitive name to relationships associated with a concept, the clause (n m) states the cardinality of a relation.

Listing 3.4: WSML formalization of PCM top concepts

```
concept Policy
    hasServiceReference ofType URI
    hasCondition ofType (0 1) PolicyCondition
    hasConditionDescription ofType (0 1) string
    hasNfp ofType (1 ∗) PolicyNfp
concept PolicyNfp
     hasExpression ofType (1 1) PolicyExpression
concept Request subConceptOf PolicyNfp
     hasRelevance ofType (1 1) _float
```

ServiceReference specifies the service(s) that the policy refers to. *PolicyCondition* defines the applicability condition of a policy. It can be specified through arbitrary axioms. However, in order to support the specification of such axioms in a structured way, the meta-model ontology provides also the concept of *Client* and the two new relations in Listing 3.5.

Listing 3.5: Relations for *PolicyCondition* specifications

```
    relation satisfiedFor (ofType PolicyCondition, ofType Client)
    relation satisfied (ofType PolicyCondition)
```

Such relations can be used to represent the head of axiom rules defining the applicability condition. The first (binary) relation can be used to relate the applicability of a *Policy* to a client. The second (unary) relation can be used to avoid a specific reference to the client. An example of exploitation of the binary relation stating that ”*the premium policy is applicable to clients subscribing for multiple travels*” is in Listing 3.6.

Listing 3.6: Example of *PolicyCondition*

```
axiom policy1ConditionAxiom
          definedBy
             applicable(premiumPolicy) :− _#[numberOfTravelsSubscribed hasValue ?n and ?n >
                 1)] memberOf LogisticOperatorClient.
```

A characteristic of PCM is the definition of properties through expressions that are formally defined by an ontology. A set of axioms define the dependencies between the three classifications of *PolicyNfp*, *NfpExpression* and *ConstraintOperator* represented in Figure 3.2.

*PolicyNfp*s are classified as *QuantitativeNfp*s and *QualitativeNfp*s. A *QuantitativeNfp* is specified by a *QuantitativeExpression* that is defined using a *QuantitativeOperator*. Conversely, a *QualitativeNfp* is specified by a *QualitativeExpression* that is defined using a *QualitativeOperator*.

An example of *QuantitativeNfp* definition is given by a *SingleValueNfp* that is specified through a *SingleValueExpression* declared using a binary operator, a numeric value and a measurement unit (see Listing 3.7).

Listing 3.7: Example of *QuantitativeNfp* definition

```
concept SingleValueNfp subConceptOf QuantitativeNfp
   hasExpression ofType (1 1) SingleValueExpression
concept SingleValueExpression subConceptOf QuantitativeExpression
   hasOperator ofType (1 1) BinaryOperator
   hasParameter ofType (1 1) _float
   hasUnit ofType (1 1) Unit
```

An example of *QualitativeNfp* definition is given by *SetNfp* that is specified through a *SetExpression* declared using a logical operator and a set of values (see Listing 3.8).

Listing 3.8: Example of *QualitativeNfp* definition

```
concept SetNfp subConceptOf QualitativeNfp
   hasExpression ofType (1 1) SetExpression
concept SetExpression subConceptOf QualitativeExpression
   hasOperator ofType (1 1) SetOperator
   hasParameters ofType (1 ∗) <anyType>
```

The specification of a *Request* differs from the one of a *PolicyNfp* for what concerns the relevance attribute. Each subConcept of *Request* inherits the relevance attribute and all the attributes of the relative *PolicyNfp*. An example is given by *SetRequest* that is a *subConceptOf Request* and *SetNfp* (see Listing 3.9).

Listing 3.9: Example of *Request* definition

```
concept SetRequest subConceptOf { SetNfp, Request }
    hasExpression ofType (1 1) SetExpression
    hasRelevance (1 1) _decimal
```

3.2 Examples of Web service contract specifications

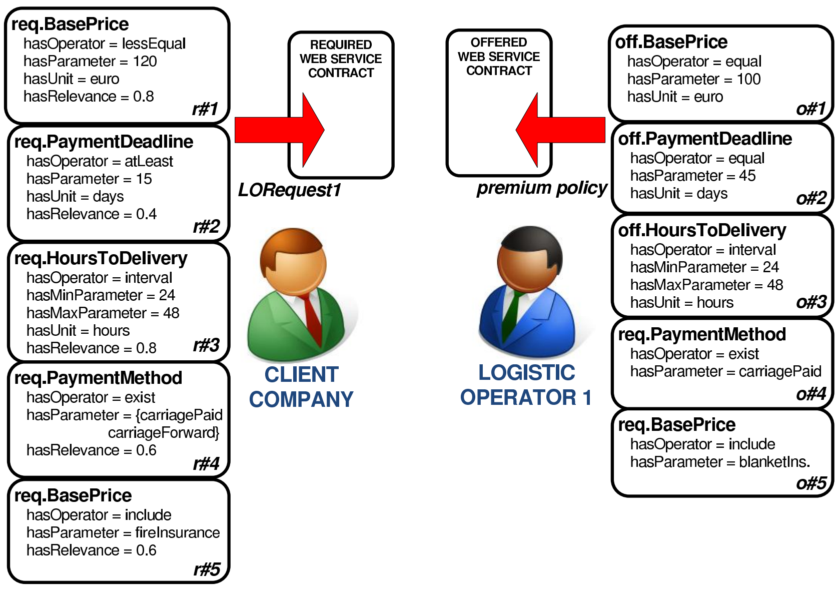


Figure 3.4: The scenario expressed in PCM terms

The FTS contract specification problem proposed in Section 2.2.1 can be solved using the PCM. The PCM-based definition of the *premiumPolicy* offered by the Logistic Operator1 is synthetically represented on the right-hand side of Figure 3.4. The PCM-based definition of the *LORequest1* is in the left-hand side of Figure 3.4.

Listing 3.10 provides the WSML description of the *premiumPolicy*. The term 'instance' is used in WSML to introduce a description of an instance of the ontology. Descriptions are flat since nested descriptions in ontology specification are not supported by current editing and processing tools for WSML. Namespaces such as ’pcm#’ and ’nfpo#’ refer respectively to the PCM ontology and a domain NFP ontology extending the PCM (see Listing B.9 in Appendix B). The namespace ’ins#’ refers to an ontology describing insurance typologies (see Listing B.10 in Appendix B).

Listing 3.10: PremiumPolicy

```
instance premiumPolicy memberOf pcm#Policy

    pcm#hasServiceReference hasValue _"http://www.example.it/WSOmniTransport#
         WSOmniTransport"
    pcm#hasConditionDescription hasValue "This policy is applicable for a client that requires multiple
         travels"
    pcm#hasCondition hasValue policyConditionAxiom

    pcm#hasNfp hasValue offBasePrice1
    pcm#hasNfp hasValue offPaymentDeadline1
    pcm#hasNfp hasValue offHoursToDeliveryRange1
    pcm#hasNfp hasValue offPaymentMethod1
    pcm#hasNfp hasValue offInsurance1

axiom policyConditionAxiom
    definedBy
        applicable(premiumPolicy) :- _#[numberOfTravelsSubscribed hasValue ?n and ?n > 1)]
             memberOf LogisticOperatorClient.


instance offBasePrice1 memberOf nfpo#BasePrice
    hasExpression hasValue offBasePriceExp1

instance offBasePriceExp1 memberOf nfpo#BasePriceExpression
    hasOperator hasValue pcm#equal
    hasParameter hasValue 100
    hasUnit hasValue nfpo#euro


instance offPaymentDeadline1 memberOf nfpo#PaymentDeadline
    hasExpression hasValue offPaymentDeadlineExp1

instance offPaymentDeadlineExp1 memberOf
nfpo#PaymentDeadlineExpression
    hasOperator hasValue pcm#equal
    hasParameter hasValue 45
    hasUnit hasValue nfpo#days


instance offHoursToDeliveryRange1 memberOf
nfpo#HoursToDeliveryRange
    hasExpression hasValue offHoursToDeliveryExp1
```

```
instance offHoursToDeliveryExp1 memberOf
nfpo#HoursToDeliveryRangeExpression
    hasOperator hasValue pcm#interval
    hasMinParameter hasValue 24
    hasMaxParameter hasValue 48
    hasUnit hasValue nfpo#hours


instance offPaymentMethod1 memberOf nfpo#PaymentMethod
    hasExpression hasValue offPaymentMethodExp1

instance offPaymentMethodExp1 memberOf
nfpo#PaymentMethodExpression
    hasOperator hasValue pcm#all
    hasParameters hasValue nfpo#carriagePaid


instance offInsurance1 memberOf nfpo#Insurance
    hasExpression hasValue offInsuranceExp1

instance offInsuranceExp1 memberOf nfpo#InsuranceExpression
    hasOperator hasValue pcm#all
    hasParameters hasValue ins#BlanketInsurance
```

Listing 3.11 provides the WSML description of the *LORequest1* requested by the Client Company.

Listing 3.11: LORequest1

```
instance LORequest1 memberOf pcm#RequestedPolicy

        pcm#hasNfp hasValue reqBasePrice
        pcm#hasNfp hasValue reqPaymentDeadline
        pcm#hasNfp hasValue reqHoursToDelivery
        pcm#hasNfp hasValue reqPaymentMethod
        pcm#hasNfp hasValue reqInsurance

instance reqBasePrice memberOf nfpo#BasePriceRequest
    hasExpression hasValue reqBasePriceExp
    hasRelevance hasValue 0.8

instance reqBasePriceExp memberOf nfpo#BasePriceExpression
    hasOperator hasValue pcm#lessEqual
    hasParameter hasValue 120
    hasUnit hasValue nfpo#euro


instance reqPaymentDeadline memberOf nfpo#PaymentDeadlineRequest
    hasExpression hasValue reqPaymentDeadlineExp
    hasRelevance hasValue 0.4

instance reqPaymentDeadlineExp memberOf
```

```
nfpo#PaymentDeadlineExpression
    hasOperator hasValue pcm#atLeast
    hasParameter hasValue 15
    hasUnit hasValue nfpo#days


instance reqHoursToDelivery memberOf nfpo#HoursToDeliveryRequest
    hasExpression hasValue reqHoursToDeliveryExp
    hasRelevance hasValue 0.8

instance reqHoursToDeliveryExp memberOf
nfpo#HoursToDeliveryExpression
    hasOperator hasValue pcm#interval
    hasMinParameter hasValue 24
    hasMaxParameter hasValue 48
    hasUnit hasValue nfpo#hour


instance reqPaymentMethod memberOf nfpo#PaymentMethodRequest
    hasExpression hasValue reqPaymentMethodExp
    hasRelevance hasValue 0.6

instance reqPaymentMethodExp memberOf nfpo#PaymentMethodExpression
    hasOperator hasValue pcm#exist
    hasParameters hasValue {nfpo#carriagePaid,nfpo#carriageForward}


instance reqInsurance memberOf nfpo#InsuranceRequest
    hasExpression hasValue reqInsuranceExp
    hasRelevance hasValue 0.6

instance reqInsuranceExp memberOf nfpo#InsuranceExpression
    hasOperator hasValue pcm#include
    hasParameters hasValue ins#FireInsurance
```

3.3 Related Works

Developing models and languages for non-functional properties has been a challenging problem in many research areas. In software engineering especially NFPs models and languages were developed to support the description of NFP aspects and requirements of software systems. In [60] for example, a language for NFPs called Process^{NFL} is proposed to express correlations and conflicts between NFPs and also compositional aspects and strengths. In [49] the ODRL-S language is proposed to describe economical and legal aspects of a service as license clauses. In [61] the CQML modeling language is proposed for specification of Quality of Service (QoS) (i.e., the particular kind of NFPs related to technological characteristics of a Web services) allowing QoS characteristics to be refined and aggregated. The QML [62] language and model is a domain independent approach to describe QoS providing a refinement mechanism that allows QoS aspects to be defined as refinements of existing ones. A formal machine processable description of software and services by means of constraints instead of ontologies is provided in [63]. In addition, QRL provides a mechanism to express alternative NFPs for the same service by means of the so-called negotiation clauses. In the Web services area some efforts to describe NFPs resulted in the WS-* specifications. Considerable related work has been done also in the context of Service Level Agreements.

Current standards for semantic descriptions of services (e.g., OWL-S [26] and WSMO [27]) only marginally cover the specification of NFPs. As discussed in Section 1.1.1, they basically adopt attribute-value descriptions.

The OWL-S approach considers the following set of NFPs: *service name*, *text description* and *quality rating*. Other non-functional properties can be included by using the *ServiceParameter* of the *ServiceProfile*. These NFPs are described in the Service Profile part and explicitly formalized using OWL.

The WSMO approach recommends a set of NFPs for each particular element of a Web service description (namely, *Goal*, *Web Service*, *Ontology* and *Mediator*). As an example, the recommended NFPs for Web service descriptions are: *accuracy, contributor, coverage, creator, date, description, financial, format, identifier, language, network-related QoS, owner, performance, publisher, relation, reliability, rights, robustness, scalability, security, source, subject, title, transactional, trust, type, version*. Using the Web Service Modeling Language (WSML), values to these NFPs can be assigned. Such values can be any identifier and thus it can be an IRI, a data value, an anonymous identifier or a list of the former. A closer look at non-functional properties used in WSML shows that many of them are used to express information about the description itself and not about the service. This point of view is different from the definition of NFP adopted by this dissertation. For example, the NFP *contributor, creator, date, identifier, owner, publisher, subject, title, and version* are not constraints on either what a service can do nor how it can do it. Another limitation of the current approach for NFPs in WSML is that these properties are not included in the logical model and thus reasoning on them is not possible.

Toma et al. [4] propose to overcome the WSMO limitation by specifying NFP through axioms. Other papers [28, 29] propose to fill the OWL-S gap by complementing it with models for Quality of Service (QoS), i.e., the subset of NFPs related to technological characteristics of a Web services. Kritikos and Plexousakis [28] propose a rich and extensible model of QoS to support matching for Web service selection. The model is designed into six facets that describe the characteristics of a QoS (e.g., metric, unit, value type). Giallonardo and Zimeo [29] define a model that allow service providers to advertise on the QoS offered, and service consumers to specify QoS requirements associated with an OWL-S profile.

Other papers (e.g., [30–32]) propose models for QoS descriptions that are not related to a particular standard. Maximilien and Singh [30] propose a model that facilitates providers in expressing policies and consumers in expressing preferences on QoS. The model is created in order to support QoS matching and matching degree evaluation. Afandi et al. [31] provide a model to extend service description and advertisement mechanisms to gather QoS information. The model provides support for Web service discovery and QoS monitoring. Tsesmetzis et al. [32] introduce a model to establish a set of rules that are used to represent QoS characteristics of Web services along with the relationships among them.

PCM addresses the following aspects for the description of NFPs: (i) the explicit distinction between requested and offered NFPs; (ii) the extensive use of constraint operators; (iii) the definition of relevance values for requested NFPs; (iv) the description of policies; (v) the definition of policy conditions. Moreover, PCM has been designed to provide a good tradeoff between expressiveness (i.e., the ability to describe the several facets related to NFPs) and complexity (i.e., the number of concepts introduced by the model). Observe that complexity influences the depth and length of a NFP description.

Table 3.1 summarizes the proposals in the literature and PCM in regards to the described aspects. Expressiveness and complexity of the approach proposed in [4] can hardly be evaluated since NFP are expressed by arbitrary NFP ontologies (e.g. the PCM itself); the use of axioms provides for flexibility at the cost of no support in writing and interpreting specifications.

Relevance and policy conditions are distinctive characteristics of the PCM. The definition of combined offers can also be considered a distinguishing characteristic of the PCM, since only Giallonardo and Zimeo and Kritikos and Plexousakis provide limited support by supporting association of more QoS offers with an OWL-S service profile. The models proposed by Afandi et al. and Tses-

	Req-Off	**Constr.Op.**	**Relevance**	**Comb.Offers**	**Conditions**	**Expressiveness**	**Complexity**
WSMO and OWL-S attribute-value [26, 27]	no	no	no	no	no	low	low
Toma et al. [4]	yes	limited	no	not explicit	yes	-	-
Kritikos and Plexousakis [28]	yes	limited	no	limited	no	high	high
Giallonardo and Zimeo [29]	yes	yes	no	limited	no	medium	medium
Maximilien and Singh [30]	not explicit	limited	no	no	no	medium	low
Afandi et al. [31]	not explicit	no	no	no	no	medium	low
Tsesmetzis et al. [32]	no	no	no	no	no	medium	low
PCM	**yes**	**yes**	**yes**	**yes**	**yes**	**medium**	**medium**

Table 3.1: Comparison of models for NFP description

metzis et al. present a good tradeoff between expressiveness and complexity but are limited w.r.t. other features in comparison. The model proposed by Maximilien and Singh provides a good balance between expressiveness and complexity but a limited set of constraint operators are supported and the explicit distinction between NFP offers and requests is not addressed. The proposal by Kritikos and Plexousakis outperforms PCM in expressiveness, but with the drawback of higher complexity due to the number of facets (with about 100 high-level concepts) that need to be described. Moreover, the specification of new constraint operators is supported but limited to numeric and string value types. The model proposed by Giallonardo and Zimeo presents similarities with our model but it does not allow an explicit distinction between NFP requests and offers. Moreover, it is strictly related to the OWL-S standard.

Chapter 4

Modeling and Mapping Web Service Contract Specifications

In the current service composition landscape there is the need to select and compose different services to provide converged services. In order to achieve an efficient service composition, for each required functionality, the best among functional-equivalent services must be selected. This activity requires the evaluation of NFPs specified in service contracts associated with each available service w.r.t. the user requirements.

Currently, service contracts are established using different approaches (e.g., policies, licenses, service level agreements) and with different specification languages, such as ODRL-S [13], WSLA [14], WSOL [33], and WS-Policy [12]. Even though these languages support the specification of similar information, there exists no shared reference ontology/thesaurus for describing service contracts. This means that service consumers and providers can specify their service contracts as they wish, raising interoperability issues due to the impossibility to perform automatic matching when multiple services referring to heterogeneous service contracts are to be processed.

This scenario underlines that NFP descriptions can be extracted from heterogeneous specifications but proper techniques are needed to make them comparable and usable to perform efficient Web service selections. Currently, no comprehensive solutions for *the Web service contract heterogeneity problem* are available in the literature. New proposals, such as the VieSLAF framework [37] and the Integrated Service Engineering (ISE) workbench [18], represent innovative but partial solutions since only the management of service level agreement (SLA) mappings is supported.

As introduced in 1.1.2, three types of service contract specification languages are available:

- *Type A*: includes *languages allowing the specification of predefined properties*. In this type, e.g., ODRL-S and WS-Policy, the properties that can be included into a specification are defined into profile models.
- *Type B*: includes *languages allowing the specification of user-defined properties*. In this type, e.g., WSLA, the properties are specified without using any shared thesaurus/ontology.
- *Type C*: includes *languages allowing the specification of properties defined in user ontologies*. In this type, e.g., WSOL, the properties are specified using external ontologies.

Different techniques must be applied for modeling and mapping terminologies of different language typologies. Ontology alignment tools [34–36], defining mappings between concepts defined in different ontologies, cannot be used to fully automate the mapping between different specifications because *Type A* and *Type B* specifications are not supposed to be based on ontologies.

This dissertation proposes a solution for *the Web service contract heterogeneity problem* that consists in using the WSML formalization of the Policy-Centered Metamodel (PCM) proposed in Chapter 3 for modeling rich NFP descriptions to be included in service contracts. Moreover, techniques to perform the mapping of existing service contracts defined in different languages to PCM-based WSML descriptions are provided in order to reuse available specifications.

This chapter is organized as follows. Section 4.1 describes the background on service contract specification languages. Section 4.2 provides an overview of the proposed solution for *the Web service contract heterogeneity problem*. Finally, Sections 4.3 and 4.4 describe the proposed techniques for modeling service contract terminology and mapping existing service contract specifications.

4.1 Web Service Contract Specification Languages

Besides functional terms, a Web service contract is composed of the specification of different types of non-functional properties, such as *Quality of Service*

(QoS), *Business*, *Service Context* and *License* terms of a service. *QoS* terms (e.g., response time) represent technical aspects of the service. *Business* terms (e.g., service price) describe financial terms and conditions. *Service Context* terms (e.g., service delivery location) define the characteristics of the context associated with the service. Finally, *License* terms (e.g., limitation of liabilities and usage permissions) state responsibilities among involved parties and conditions on service usage.

Web service contracts can be can be established using different approaches and described using different specification languages. This section presents details about the most common Web service contract specification languages.

4.1.1 The Open Digital Rights Language for Services

The Open Digital Rights Language for Services (ODRL-S) [13] is an XML-based language designed to describe licensing clauses of a service in machine interpretable form. A service could allow/deny itself to be used/accessed by other services with or without modification. For drafting machine readable licenses, the clauses of a service license should be unambiguous. ODRL-S specifications can include only clauses that are formalized in the ODRL-S Profile. This profile formalizes service licensing clauses in terms of *subject, scope of rights, financial terms, warranties, indemnities, limitations of liability* and *evolution* of the service.

The *subject* of the license relates to the definition of the service being licensed. This defines some related information about the service and may include a unique identification code for the service, service name, service location, and other relevant information.

The *scope of rights* of a service license reflect on what can be done with the service. The ODRL-S Profile defines permissions that are directly related to the service usage:

- *Adaptation*: refers to the right of allowing the use of service interface only (i.e., execution).
- *Composition*: refers to the right of execution with the right of interface modification. Composition is dependent on the execution of services being composed.
- *Derivation*: refers to the right of allowing modifications to the service interface as well as the implementation of a service.

- *Attribution*: states that a service may expect the attribution for its use by another service in any of the forms.
- *ShareAlike*: states that any derivative of the service must be licensed under the same terms as the original license.
- *Non-Commercial Use*: refers to the right of using the service for non-commercial or for commercial purposes.

The ODRL-S Profile includes elements allowing the description of the following *financial terms* and conditions:

- *Prepay*: a lump sum payment can be paid by the service consumer before using the service.
- *Postpay*: a lump sum payment can be paid after using the service.
- *Peruse*: a lump sum payment must be paid for using the service.

The *Warranties* of a service license regard the description of QoS offered by the service provider. Warranties are further categorized into:

- *Performance*: QoS based on temporal dimensions, such as:
  - *Mean Response Time*: the mean time between the moment a request is sent to the time that the response has been provided to the consumer.
  - *Mean Process Time*: the average time for processing a service.
  - *Mean Latency*: the average round-trip time between sending a request and receiving the response.
  - *Mean Throughput*: measure of the average amount of service that is provided.
- *Compliance*: quality aspects of the service in conformance with the rules, the law and compliance with standards, such as:
  - *Resolution Rate*: the average time for resolving problems related to service provisioning.
- *Reliability*: technical measures related to the service configuration and network connections, including:

  - *Max Utilization*: the performance measure of a service related by throughput and response time.
  - *Mean Availability Rate*: the probability of the service being accessible.
  - *Mean Service Level Violation Rate*: the mean occurrences of breach of SLA clauses.

- *Monitoring*: measures that support the management of services, such as:
  - *Down Time Frequency*: the frequency at which a service provider verifies the availability of a service.

The *Indemnities* of a service license specify the provision of defense by the licensor for the licensee if a third party sues the licensee, alleging that the licensee's use of the licensed software infringes or violates the third party's intellectual property rights.

The *Limitation of liability* clauses restrict the liability of each of the parties under the license agreement, including:

- *Network Errors*: the licensor will not be liable for problems with the network.
- *Infrastructure Errors*: the licensor will not be liable for problems caused by the infrastructure.

Finally, the *Evolution* clauses of a service license specify functional and/or non-functional modifications that the service provider can perform, and takes the form of new releases or new versions. *Evolution* clauses are:

- *Max Upgrades*: the allowed number of upgrades to a service before the license becomes invalid.
- *Max Versions*: the allowed number of versions of a service before the license becomes invalid.
- *Substitutable*: the service can be substituted for by another similar service of same scope to which the licensee is allowed to use.
- *Generic*: the service can be replaced by a service of broader scope and the licensee retains the rights over the generic service.

4.1.2 The Web Service Level Agreement language

The Web Service Level Agreement (WSLA) language [14] is used by both service providers and service consumers to define service performance characteristics and the way to evaluate and measure them. WSLA specifications comprise:

- A description of the *parties*, their roles (provider, customer, third parties) and the action interfaces they expose to the other parties of the contract.
- A *service definition* containing detailed specification of the *SLA parameters* (e.g., service response time) guarantees are applied to.
- A representation of the *parties' obligations* defined as *Service level objectives* containing formal expressions of the guaranteed conditions of a service in a given period.

SLA parameters are the main elements of a service description. A parameter describes an observable property of a service whose value can be obtained from a source of measurement. Each *SLA Parameter* is defined by mean of:

- a *metric* element: how the value of the SLA parameter is computed. This metric is used to determine the SLA parameter's value at execution time.
- an attribute *name*: a unique ID of the parameter.
- an attribute *type*: the domain of the metric's value.
- an attribute *unit*: the unit of measurement of the metric's value.
- a *communication* element: how other parties get access to values of the SLAParameter. The communication section can be omitted if no interaction with other parties is expected.

Figure 4.1 shows examples of SLAParameter specifications. The SLA parameter named *Price* assumes a value of type *float* and unit equal to *euro*. It is based on the metric *Usage*. The SLA parameter named *TransactionLimit* assumes a value of type *float* and unit equal to *query / day*. It is based on the metric *Transactions*. The party *ACMEProvider* is commissioned to compute the value of *RequestLimit* and it must send all new values to the party *ZAuditing*. In addition, *ZAuditing* may retrieve the value (pull), too.

```
<?xml version="1.0" encoding="UTF-8"?>
...
<SLAParameter name=”Price”
                    type=”float”
                    unit=”euro”>
    <Metric>Usage</Metric>
</SLAParameter>
<SLAParameter name=”TransactionLimit”
                    type=”float”
                    unit=”query / day”>
    <Metric>Transactions</Metric>
    <Communication>
        <Source>ACMEProvider</Source>
        <Pull>ZAuditing</Pull>
        <Push>ZAuditing</Push>
    </Communication>
</SLAParameter>
```

Figure 4.1: Examples of SLA Parameter specifications

Service level objectives defines the state that the service must guarantee as an *Expression* on predicates that refer to defined *SLA Parameters*. In particular, a *Service level objectives* is defined by mean of:

- an *obliged* element: a reference to a party that is in charge of delivering what is promised in this obligation.
- a *validity period*: when the guarantee is applicable.
- an *expression*: the actual content of the obligation, that is, what is asserted by the service provider to the service customer.
- an *evaluation event* or a *schedule* element: the case in which or the schedule according to the expression of the service level objective is to be evaluated.
- a *name*: a unique ID assigned to the guarantee.

Each *expression* follows first order logic, including predicates and logic operators. The simplest form of a logic expression is a plain predicate. The conjunction AND or the disjunction OR are used to connect two logic expressions. The unary NOT is used to negate an expression. The implication IMPLIES is equivalent to the logic subjunction (A IMPLIES B is the same as NOT A OR B).

```
<?xml version="1.0" encoding="UTF-8"?>
...
<Expression>
  <And>
    <Expression>
      <Predicate xsi:type=equal>
        <SLAParameter>Price</SLAParameter>
          <Value>9.99</Value>
      </Predicate>
    </Expression>
    <Expression>
      <Predicate xsi:type="Less">
        <SLAParameter>TransactionLimit</SLAParameter>
          <Value>100</Value>
      </Predicate>
    </Expression>
  </And>
</Expression>
```

Figure 4.2: Examples of Service Level Objective specifications

Figure 4.2 shows an example of *Service level objective expression* specified using the SLA parameters defined in the example in Figure 4.1. The expression states that the price of the service is equal to 9.99 euro and the number of accepted transactions in a day must be less than 100.

4.1.3 The Web Service Offerings Language

The Web Service Offerings Language (WSOL) [33] is an XML notation fully compatible with the standard WSDL. The central concept in WSOL is a class of service. Classes of service of one Web Service refer to the same WSDL description, but differ in constraints and management statements. In WSOL, the term *service offering* is used to refer to the formal representation of a single class of service of one Web Service. Consequently, a WSOL Service Offering contains formal representations of various constraints and management statements that determine the corresponding class of service. A service offering is obtained by the formal specification of constraints and statements.

The constraints could be of the following different types:

- *Functional constraints*: referring to some conditions that need to be satisfied for the execution of a Web Service operation to be functionally correct. WSOL supports specification of several functional constraints such as pre-conditions, post-conditions, and future-conditions.

- *QoS constraints*: referring to one or more QoS of an operation invocation such as *performance, reliability, availability, response time*, etc.
- *Access rights*: referring to a condition under which a consumer using a particular service offering has the right to access a particular operation. Access is allowed only if an access right constraint for the operation is explicitly specified and satisfied.

The statements enable the specification of important service information. Examples of statement are the following:

- *Price*: the amount of money that a Web service consumer has to pay to a supplier Web Service for using a particular operation of the Web Service.
- *Penalty*: the amount of money that a supplier Web Service has to pay to a Web Service consumer if it could not fulfil the constraints.

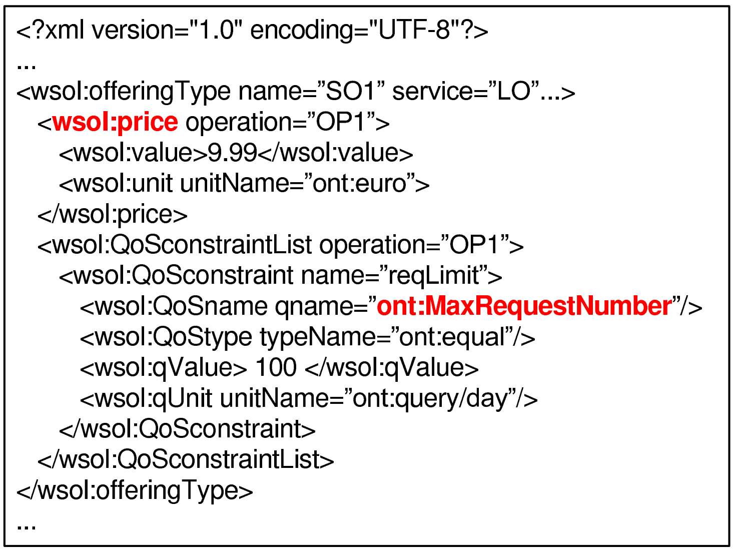
Figure 4.3 shows a fragment of a WSOL specification. The *service offering SO1* associated with *service LO* defines a statement on *price* (i.e., the price for the usage of *operation OP1* is equal to 9.99 Euro) and a QoS constraint (i.e., the service accepts a maximum of 100 requests in a day for the *operation OP1*). Note that the meaning of maxRequestNumber, equal, and query/day is defined in an external ontology (namespace ont). This is a important characteristic of the WSOL. Service providers and consumers are free to use their own terminology inside a WSOL specification but this terminology is represented into external ontologies.

4.1.4 The Web Service Policy Language

The Web Service Policy (WS-Policy) language [12] offers mechanisms to represent the capabilities and requirements of Web services as policies.

A *policy* is defined as a potentially empty collection of *policy alternatives* that is a potentially empty collection of policy assertions. A *policy assertion* represents a requirement, a capability, or other property of a service. A *policy scope* is a collection of *policy subjects* (e.g., an endpoint, message, resource, operation) with which a policy can be associated.

Policy assertions can be combined in different ways to express consistent combinations of behaviors (capabilities and requirements). There are three policy operators for combining policy assertions: *Policy*, *All* and *ExactlyOne*.



```
<?xml version="1.0" encoding="UTF-8"?>
...
<wsol:offeringType name=”SO1” service=”LO”...>
  <wsol:price operation=”OP1”>
    <wsol:value>9.99</wsol:value>
    <wsol:unit unitName=”ont:euro”>
  </wsol:price>
  <wsol:QoSconstraintList operation=”OP1”>
    <wsol:QoSconstraint name=”reqLimit”>
      <wsol:QoSname qname=”ont:MaxRequestNumber”/>
      <wsol:QoStype typeName=”ont:equal”/>
      <wsol:qValue> 100 </wsol:qValue>
      <wsol:qUnit unitName=”ont:query/day”/>
    </wsol:QoSconstraint>
  </wsol:QoSconstraintList>
</wsol:offeringType>
...
```

Figure 4.3: An example of WSOL specification

Combining policy assertions using the *Policy* or *All* operator means that all the behaviors represented by these assertions are required. Combining policy assertions using the *ExactlyOne* operator means that exactly one of the behaviors represented by these assertions is required. Policy assertions can be marked as optional or ignorable. The *Optional* attribute is used to represent behaviors that may be engaged for an interaction. The *Ignorable* attribute is used by service providers to clearly indicate which policy assertions refer to behaviors that do not manifest on the wire and may not be of concern to a requester when determining policy compatibility.

The WS-Policy language provides only mechanisms to represents policies. Policy assertions refer to external models. An example is the WS-SecurityPolicy [64] that defines a set of security policy assertions that have been designed to work independently of a specific version of WS-Policy. Flexibility with respect to token types, cryptographic algorithms and mechanisms used, including using transport level security is part of the WS-SecurityPolicy specification. The intent is to provide enough information for compatibility and interoperability to be determined by Web service participants along with all information necessary to actually enable a participant to engage in a secure exchange of messages.

Figure 4.4 shows an example of WS-Policy specification indicating a Transport Binding that includes a Transport Token and an Algorithm Suite. The

```
<wsp:Policy xmlns:wsp="..." xmlns:wsp="...">
  <sp:TransportBinding>
      <wsp:All>
          <sp:TransportToken>
              <wsp:All>
                  <sp:HttpsToken />
              </wsp:All>
          </sp:TransportToken>
          <sp:AlgorithmSuite>
              <wsp:All>
                  <sp:Basic256 />
              </wsp:All>
          </sp:AlgorithmSuite>
      </wsp:All>
  </sp:TransportBinding>
</wsp:Policy>
```

Figure 4.4: An example of WS-Policy specification

assertions regarding Transport Token, Algorithm Suite and their values (i.e., HttpsToken and Basic256) are defined in the WS-SecurityPolicy specification. HttpsToken represents a requirement for a transport binding to support the use of HTTPS. Basic256 represents the algorithm suite required for performing cryptographic operations.

4.2 Solving Web Service Contract Heterogeneity

Several actors are involved in the management of Web service contracts, such as:

- *Service Providers*: offer services characterized by contracts specified in different languages. Service providers are also involved in contract negotiation with service consumers.

- *Service Consumers*: submit a request for service selection/composition specifying a contract including functional and non-functional requirements.

- *Language Experts*: possess the knowledge of the profile models of languages (e.g., ODRL-S, WSLA, WSOL, WS-Policy) for the specification of service contracts.

- *Domain Experts*: possess the knowledge of the terminology used into a particular domain.

In order to allow the identified actors to manage heterogeneous service contract specifications, a reference ontology must be defined. The *Service Contract (SeCO) Reference Ontology* provides a PCM-based description of properties that can be included in a service contract and it is composed of:

- a *core part* containing the specification of common properties (e.g., QoS);
- a *plug-in part* that can be enriched by Language Experts with new properties.

The definition of the *SeCO Reference Ontology* and its usage for mapping existing Web service contracts require to identify techniques for: (i) *modeling* service contract properties as SeCO ontological concepts; (ii) *mapping* native service contract properties (e.g., properties included into the ODRL-S profile model) to SeCO ontological concepts.

Different techniques must be applied by Language Experts for modeling and mapping terminologies of different language typologies into the *SeCO Reference Ontology*.

Languages in *Type A* (e.g., ODRL-S and WS-Policy/WS-Security) are characterized by profile models describing all the properties that can be included in a service contract. In this case, Language Experts enrich the plug-in part modeling all the properties not included in the reference ontology. Moreover, Language Experts can define fixed mapping rules between native properties and ontological concepts.

Languages in *Type B* (e.g., WSLA) allow new properties to be defined. This characteristic limits the possibility to perform the modeling and mapping of properties in advance. Thus, interactions to the Service Providers are still needed. However, users of the same domain (e.g., the logistic operator domain) typically utilize common terminologies, e.g., logistic operator service providers utilize the term *Shipping Location* in their specifications which has the same meaning of the property *Service Delivery Location* available in the core part of the SeCO Reference Ontology. Moreover, the users of a domain can decide to use their native language for describing service properties. As an example, the users of an Italian company can use the term *Tempo di Risposta* in their specifications to indicate the property *Response Time*. Common terminologies and domain-specific knowledge are used by Language Experts and Domain Experts to define customized mapping rules which will reduce the interactions needed to perform the mapping.

Languages in *Type C* (e.g., WSOL) are similar to the ones in *Type B* but here the properties are semantically described in external ontologies. Thus, the possibility to perform the modeling and mapping activities is limited and the definition of customized mapping rules between concepts in the most common user ontologies and SeCO ontological concepts is required.

4.3 Modeling Web Service Contract Terminology

Automatic techniques for modeling service contract properties as SeCO ontological concepts cannot be defined. The necessity to interpret the nature of the property and to define relations (e.g., similarity) with already modelled concepts requires the knowledge provided by Language Experts. However, when an XML-based profile model defining properties is available (i.e., Languages in *Type A*), a set of general rules can support Language Experts to extract properties from the profile model and semantically describe them into the SeCO Reference Ontology. General rules link an XML-structure to a proper PCM-based description. The same structure can be associated with several rules because also the nature of the property must be considered.

Figure 4.5 shown examples of general rules linked to XML-structures. The XML-structure in which an element *C1* has a set of sub-elements (e.g., *C2* and *C3*) can be linked to three different rules:

- *InstanceOf*-rule: the sub-elements are considered as possible values assumed by *C1*.
- *Split*-rule: the sub-elements are described separately and the relationship with *C1* is omitted.
- *Split & Merge*-rule: the relationship with *C1* is omitted but the sub-elements are aggregated under a different concept.

The XML-structure in which different elements have the same sub-element *C1* is linked to the following rule:

- *IsA*-rule: the elements (e.g., *C2* and *C3*) with the same sub-element (e.g., *C1*) are considered as specializations of the sub-element.

XML Structures	Rules	SeCO Reference Ontology
<C1> <C2/> <C3/> <C1/>	*"InstanceOf"*	***concept*** *C1* ***subConceptOf*** *pcm#PolicyNfp* *pcm#hasParameters* ***impliesType*** *ConceptValue* ***instance*** *c2* ***memberOf*** *ConceptValue* ***instance*** *c3* ***memberOf*** *ConceptValue*
	"Split"	***concept*** *C2* ***subConceptOf*** *pcm#PolicyNfp* ***concept*** *C3* ***subConceptOf*** *pcm#PolicyNfp*
	"Split & Merge"	***concept*** *NewC* ***subConceptOf*** *pcm#PolicyNfp* *pcm#hasParameters* ***impliesType*** *NewCValue* ***instance*** *c2* ***memberOf*** *NewCValue* ***instance*** *c3* ***memberOf*** *NewCValue*
<C2> <C1/> <C2/> <C3> <C1/> <C3/>	*"IsA"*	***concept*** *C1* ***subConceptOf*** *pcm#PolicyNfp* ***concept*** *C2* ***subConceptOf*** *C1* ***concept*** *C3* ***subConceptOf*** *C1*

Figure 4.5: Examples of modeling rules

To illustrate the above-mentioned rules, examples related to modeling and mapping ODRL-S terminology are proposed. Figure 4.6 shows how general rules can be used to model ODRL-S properties into the SeCO Reference Ontology. The following ODRL-S terms (see Section 4.1.1) are considered:

- *Adaptation*, *Composition* and *Derivation*;
- *PrePay* and *PostPay*;
- *Mean Response Time*;
- *Attribution* and *ShareAlike*.

In the ODRL-S profile model, *Adaptation*, *Composition* and *Derivation* are sub-elements of *Permission*. For this property the ODRL-S Language Expert uses the *InstanceOf*-rule to define a new concept *Permissions* in the ontology which can assume a fixed set of values (i.e., *pcm#hasParameters impliesType PermissionValue*) that are *Adaptation*, *Composition* and *Derivation*.

PrePay and *PostPay* are super-elements for *Payment*. In this case the *IsA*-rule is applied considering them as specializations of the term *Payment*. A new

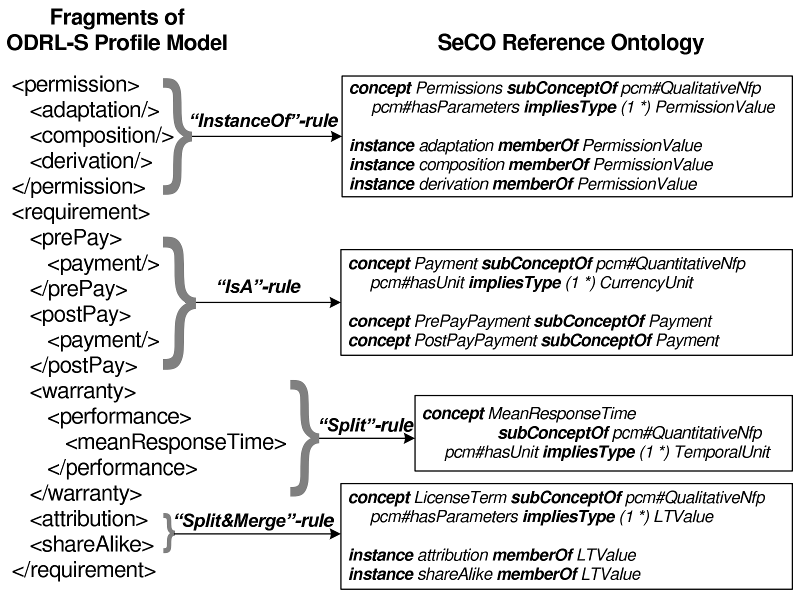

Fragments of ODRL-S Profile Model
SeCO Reference Ontology
<permission>
<adaptation/>
<composition/>
<derivation/>
</permission>
<requirement>
<prePay>
<payment/>
</prePay>
<postPay>
<payment/>
</postPay>
<warranty>
<performance>
<meanResponseTime>
</performance>
</warranty>
<attribution>
<shareAlike>
</requirement>
"InstanceOf"-rule
"IsA"-rule
"Split"-rule
"Split&Merge"-rule
concept Permissions subConceptOf pcm#QualitativeNfp pcm#hasParameters impliesType (1 *) PermissionValue
instance adaptation memberOf PermissionValue
instance composition memberOf PermissionValue
instance derivation memberOf PermissionValue
concept Payment subConceptOf pcm#QuantitativeNfp pcm#hasUnit impliesType (1 *) CurrencyUnit
concept PrePayPayment subConceptOf Payment
concept PostPayPayment subConceptOf Payment
concept MeanResponseTime subConceptOf pcm#QuantitativeNfp pcm#hasUnit impliesType (1 *) TemporalUnit
concept LicenseTerm subConceptOf pcm#QualitativeNfp pcm#hasParameters impliesType (1 *) LTValue
instance attribution memberOf LTValue
instance shareAlike memberOf LTValue

Figure 4.6: Modeling ODRL-S terms in the SeCO Reference Ontology

concept *Payment* and two sub-concepts (*PrePayPayment* and *PostPayPayment*) are added to the ontology.

Mean Response Time is sub-elements of *Performance*, *Warranty* and *Requirement*. These relationships are considered negligible and the *Split*-rule is applied. A new concept *MeanResponseTime* is added to the ontology.

Attribution and *ShareAlike* are sub-elements of *Requirement*. The *Split & Merge*-rule is used to split the relationship with *Requirement* and to consider these terms as possible values of a new concept (*LicenseTerm*).

4.4 Mapping Web Service Contract Specifications

After the modeling of properties into the SeCO Reference Ontology, Language Experts define *mapping rules* between the native properties (e.g., terms in XML-based profile models) and the defined ontological concepts. These mapping rules between terms support the mapping of existing service contracts to PCM-based service contract specifications. According to the PCM terminology, in the following PCM-based Web service contract specifications defined using the *SeCO Reference Ontology* will be called *SeCO Policies*.

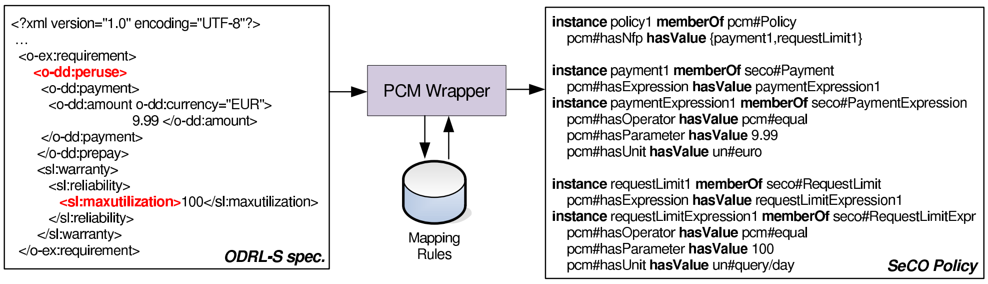
Mapping rules are applied by a PCM Wrapper in order to perform the mapping from service contract specifications to SeCO Policy. A proper technique for each type of language is required. The mapping of specifications in *Type A* language is directly performed by applying the mapping rules defined by Language Experts. For what concern specifications in *Type B* and *Type C* languages the mapping activity may require interactions with the Service Providers to handle the absence of knowledge (i.e., mapping rules) on specified properties. The Service Providers must define the mapping between their properties (i.e., text labels for *Type B* and ontological concepts for *Type C*) and concepts available in the SeCO Reference Ontology.

This section will show how a service contract stating that

- a payment of 9.99 Euros is needed to use the service;
- the service accepts a maximum of 100 requests in a day

can be mapped to a SeCO Policy. The cases in which the service contract is specified using ODRL-S, WSLA and WSOL are considered.

The mapping of the ODRL-S service contract is automatic performed using the *SeCO Reference Ontology* and predefined *mapping rules*.



```
<?xml version="1.0" encoding="UTF-8"?>
 …
  <o-ex:requirement>
     <o-dd:peruse>
       <o-dd:payment>
         <o-dd:amount o-dd:currency="EUR">
                       9.99 </o-dd:amount>
       </o-dd:payment>
      </o-dd:prepay>
      <sl:warranty>
         <sl:reliability>
            <sl:maxutilization>100</sl:maxutilization>
         </sl:reliability>
      </sl:warranty>
  </o-ex:requirement>
```

```
instance policy1 memberOf pcm#Policy
   pcm#hasNfp hasValue {payment1,requestLimit1}

instance payment1 memberOf seco#Payment
   pcm#hasExpression hasValue paymentExpression1
instance paymentExpression1 memberOf seco#PaymentExpression
   pcm#hasOperator hasValue pcm#equal
   pcm#hasParameter hasValue 9.99
   pcm#hasUnit hasValue un#euro

instance requestLimit1 memberOf seco#RequestLimit
   pcm#hasExpression hasValue requestLimitExpression1
instance requestLimitExpression1 memberOf seco#RequestLimitExpr
   pcm#hasOperator hasValue pcm#equal
   pcm#hasParameter hasValue 100
   pcm#hasUnit hasValue un#query/day
```

Figure 4.7: Mapping between ODRL-S specification and SeCO Policy

Figure 4.7 shows that the SeCO Policy is created by a PCM Wrapper with a procedure based on the following steps:

- parse the ODRL-S specification in order to detect properties;
- retrieve the mapping rules related to the detected properties;
- apply the mapping rules.

For what concern the considered example, the mapping rules to be applied are the following: (i) the ODRL-S term *peruse* must be mapped to an instance of the SeCO concept *Payment*. The currency EUR must be mapped to the term *Euro* defined in an external ontology; (ii) the ODRL-S term *maxutilization* must be mapped to an instance of the SeCO concept *RequestLimit* characterized by a unit equal to term *query/day* defined in an external ontology.

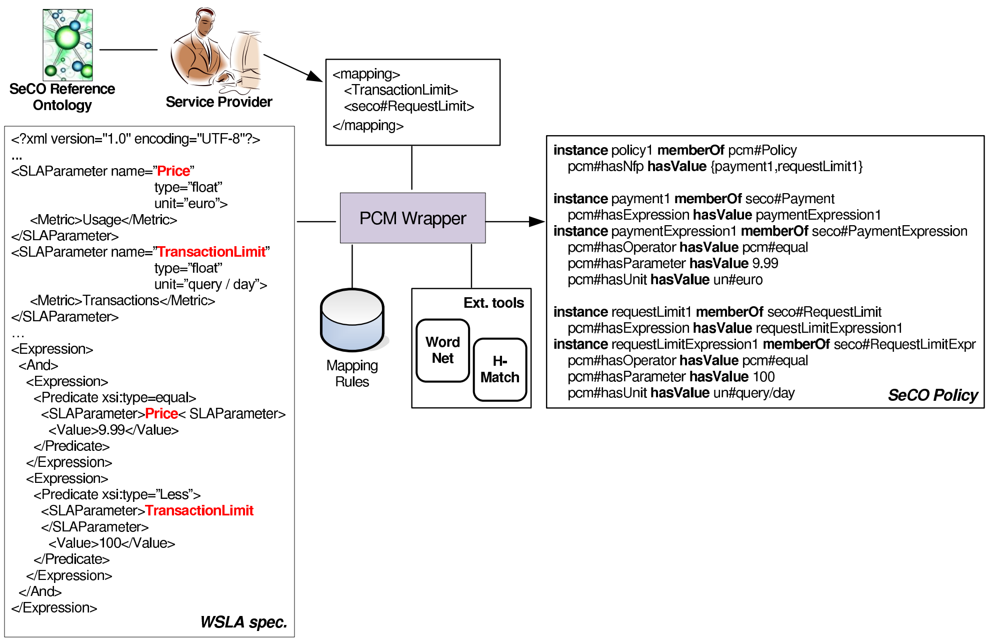
The final result for the considered example is a *Policy* containing: (i) an instance of *Payment* stating that the amount to be paid for using the service is equal to 9.99 Euros (i.e., pcm#hasOperator hasValue pcm#equal; pcm#hasParameter hasValue 9.99; pcm#hasUnit hasValue un#euro) and (ii) an instance of *RequestLimit* characterized by an expression stating that the maximum of accepted queries in a day is equal to 100 (i.e., pcm#hasOperator hasValue pcm#equal; pcm#hasParameter hasValue 100; pcm#hasUnit hasValue un#query/day). Notice that the pcm# namespace is used to refer to concepts defined in the PCM, seco# is used to refer to concepts defined in the SeCO reference ontology and un# is used to refer to concepts defined in an external ontology specifying unit of measurements.

For what concern specifications in *Type B* languages, lexical databases like WordNet can be used to support Service Providers to define mapping rules identifying synonyms between text labels and ontological concepts defined in the SeCO Reference Ontology.

The procedure used to perform the mapping is the following:

- parse the specification in order to detect properties (i.e., *SLAParameters*);
- search the availability of customized mapping rules related to the detected properties;
- if mapping rules are not identified, use WordNet to identify a possible mapping between the *SLAParameters* and concepts available in the SeCO Reference Ontology and ask confirmation about the correctness of the mapping to the Service Provider;
- if the mapping is not correct or not available, ask to the Service Provider to perform the mapping manually.

Figure 4.8 illustrates the above-mentioned steps when mapping a WSLA-based service contract defined on the SLAParameters *Price* and *TransactionLimit*. In this example, a customized mapping rule for *Price* is identified. On the contrary, the term *TransactionLimit* is not known and no rules are available. Moreover, no synonym relations are specified in WordNet between *TransactionLimit* and terms defined in the SeCO Reference Ontology. In order to handle this absence of knowledge, the Service Provider is asked to navigate the ontology and map the SLAParameter to any ontological concept. The result is the mapping of *TransactionLimit* with the concept *RequestLimit*.



```
<mapping>
  <TransactionLimit>
  <seco#RequestLimit>
</mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
...
<SLAParameter name="Price"
                     type="float"
                     unit="euro">
   <Metric>Usage</Metric>
</SLAParameter>
<SLAParameter name="TransactionLimit"
                     type="float"
                     unit="query / day">
   <Metric>Transactions</Metric>
</SLAParameter>
...
<Expression>
 <And>
  <Expression>
   <Predicate xsi:type=equal>
    <SLAParameter>Price< SLAParameter>
     <Value>9.99</Value>
   </Predicate>
  </Expression>
  <Expression>
   <Predicate xsi:type="Less">
    <SLAParameter>TransactionLimit
    </SLAParameter>
     <Value>100</Value>
   </Predicate>
  </Expression>
 </And>
</Expression>
```

```
instance policy1 memberOf pcm#Policy
   pcm#hasNfp hasValue {payment1,requestLimit1}

instance payment1 memberOf seco#Payment
   pcm#hasExpression hasValue paymentExpression1
instance paymentExpression1 memberOf seco#PaymentExpression
   pcm#hasOperator hasValue pcm#equal
   pcm#hasParameter hasValue 9.99
   pcm#hasUnit hasValue un#euro

instance requestLimit1 memberOf seco#RequestLimit
   pcm#hasExpression hasValue requestLimitExpression1
instance requestLimitExpression1 memberOf seco#RequestLimitExpr
   pcm#hasOperator hasValue pcm#equal
   pcm#hasParameter hasValue 100
   pcm#hasUnit hasValue un#query/day
```

Figure 4.8: Mapping between WSLA specification and SeCO Policy

After this preliminary step, the mapping proceeds considering the *Expressions* defined in each *Service Level Objective* of the WSLA specification. Each *Expression* follows first order logic, including predicates and logic operators. According to the logic operators, different mapping rules can be applied.

As shown in Figure 4.9 the simplest form of a logic expression is a plain predicate. The mapping to a SeCO Policy specification is executed as follow: (i) the *SLAParameter* is used to identify in the reference ontology the related concept specified by the *Service Provider*; (ii) a new instance of this concept is created. It must be characterized by an expression having constraint operator and parameter equals to *Type* and *Value* of the *Service Level Objective*; (iii) a new *Policy* containing the concept instance is created.

The logic operator *And* supports the aggregation of two or more plain predicates. The mapping to a SeCO Policy consists in defining a *Policy* containing the concept instances related to all the plain predicates. The logic operator *Or* allows the aggregation of alternative *SLAParameter* defined as plain predicates. The mapping to a SeCO Policy consists in defining a *Policy* for each

Logic Expr.	**WSLA spec.**	**SeCO Policy spec.**

Plain Predicate	<Predicate xsi:type="*TYPE*">
	  <SLAParameter>*SLAPar*</SLAParameter>
	  <Value>*VALUE*</Value>
	</Predicate>
		instance policy **memberOf** pcm#Policy
		  ...
		  hasNfp **hasValue** *SLAPar*
		
		instance *SLAPar* **memberOf** sro#*SLAPar*
		  pcm#hasExpression **hasValue** SLAParExpr
		instance SLAParExpr **memberOf** sro#SLAParExpr
		  pcm#hasOperator **hasValue** *TYPE*
		  pcm#hasParameters **hasValue** *VALUE*

And	<Expression>
	  <And>
	    <Expression>
	      --- *Plain Predicate 1* ---
	    </Expression>
	    <Expression>
	      --- *Plain Predicate 2* ---
	    </Expression>
	  </And>
	</Expression>
		instance policy **memberOf** pcm#Policy
		  ...
		  hasNfp **hasValue** *SLAPar1*
		  hasNfp **hasValue** *SLAPar2*
		  ...

Or	<Expression>
	  <Or>
	    <Expression>
	      --- *Plain Predicate 1* ---
	    </Expression>
	    <Expression>
	      --- *Plain Predicate 2* ---
	    </Expression>
	  </Or>
	</Expression>
		instance policyA **memberOf** pcm#Policy
		  ...
		  hasNfp **hasValue** *SLAPar1*
		  ...
		
		instance policyB **memberOf** pcm#Policy
		  ...
		  hasNfp **hasValue** *SLAPar2*
		  ...

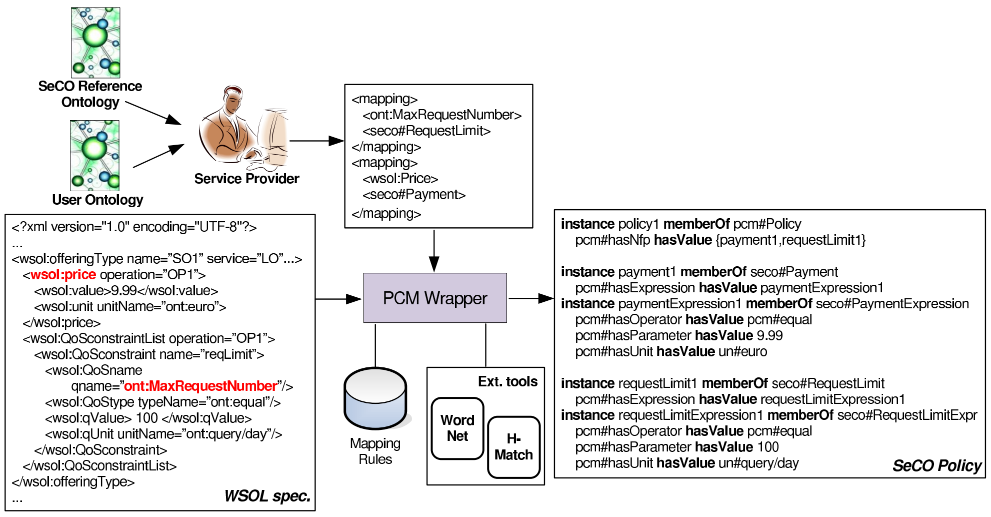
Implies	<Expression>
	  <Implies>
	    <Expression>
	      --- *Plain Predicate 1* ---
	    </Expression>
	    <Expression>
	      --- *Plain Predicate 2* ---
	    </Expression>
	  </Implies>
	</Expression>
		instance policy **memberOf** pcm#Policy
		  ...
		  hasCondition **hasValue** ---*Plain Predicate 1*---
		  hasNfp **hasValue** *SLAPar2*
		  ...

Figure 4.9: Mapping of WSLA expressions

alternative plain predicates. The mapping of the logic operator *Implies* is more complicated. This operator permits the definition of several plain predicated and the first of them is the condition for the others. For this reason the mapping is performed creating a new *Policy* containing: (i) an applicability condition defined by the first plain predicate and (ii) the concept instances related to all the plain predicates except the first.

In the example shown in Figure 4.8, the logic operator *And* is detected and the related mapping rule is retrieved. The rule is applied using the correspondences between *SLAParameters* and ontological concepts specified by the *Service Provider* in the preliminary step. The final result is the equal to the one obtained for the wrapping of a ODRL-S specification.

This approach to define mapping between WSLA specifications and SeCO Policy is used to solve the example of *service contract heterogeneity problem* described in Section 2.2.2. The WSLA specification in Listing 2.1 can be mapped to the *silver policy* shown in Listing A.2 in Appendix A.



```
<mapping>
  <ont:MaxRequestNumber>
  <seco#RequestLimit>
</mapping>
<mapping>
  <wsol:Price>
  <seco#Payment>
</mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
...
<wsol:offeringType name="SO1" service="LO"...>
  <wsol:price operation="OP1">
    <wsol:value>9.99</wsol:value>
    <wsol:unit unitName="ont:euro">
  </wsol:price>
  <wsol:QoSconstraintList operation="OP1">
    <wsol:QoSconstraint name="reqLimit">
      <wsol:QoSname
          qname="ont:MaxRequestNumber"/>
      <wsol:QoStype typeName="ont:equal"/>
      <wsol:qValue> 100 </wsol:qValue>
      <wsol:qUnit unitName="ont:query/day"/>
    </wsol:QoSconstraint>
  </wsol:QoSconstraintList>
</wsol:offeringType>
...
```

```
instance policy1 memberOf pcm#Policy
  pcm#hasNfp hasValue {payment1,requestLimit1}

instance payment1 memberOf seco#Payment
  pcm#hasExpression hasValue paymentExpression1
instance paymentExpression1 memberOf seco#PaymentExpression
  pcm#hasOperator hasValue pcm#equal
  pcm#hasParameter hasValue 9.99
  pcm#hasUnit hasValue un#euro

instance requestLimit1 memberOf seco#RequestLimit
  pcm#hasExpression hasValue requestLimitExpression1
instance requestLimitExpression1 memberOf seco#RequestLimitExpr
  pcm#hasOperator hasValue pcm#equal
  pcm#hasParameter hasValue 100
  pcm#hasUnit hasValue un#query/day
```

Figure 4.10: Mapping between WSOL specification and SeCO Policy

For what concern specifications in *Type C*, different ontology alignment tools can be also used to support the mapping of specifications in *Type C* languages: (i) tools for defining a mapping between concepts in two different ontologies by finding pairs of related concepts (e.g., *ANCHORPROMPT* [34]) or by evaluating semantic affinity between concepts (e.g., *H-MATCH* [35]) and (ii) tools for defining mapping rules to relate only relevant parts of the source ontologies (e.g., *ONION* [36]).

It is quite simple to identify a mapping between WSOL and SeCO Policy specifications. The concept of *service offering* in WSOL can be mapped to a *policy* in a SeCO Policy specification. *QoS constraints* and *statements* can be represented as *PolicyNfps*. Finally, *access rights* can considered as *Policy conditions*.

Figure 4.10 shows how the mapping of the WSOL specification is performed. Also in this case a preliminary step is needed. The procedure used is the following:

- parse the specification in order to detect properties (i.e., *QoSConstraints* and *statements*);
- search the availability of customized mapping rules related to the detected properties;

- if mapping rules are not identified, use external ontology alignment tools to identify a possible mapping between the properties defined in the user ontology and concepts available in the SeCO Reference Ontology. Ask confirmation about the correctness of the mapping to the Service Provider;
- if the mapping is not correct or not available, ask to the Service Provider to perform the mapping manually.

After this preliminary step, the mapping proceeds defining a new instance for each corresponding SeCO concept. If it is related to a *QoSConstraint*, the instance must be characterized by an expression having operator, parameter and unit equals to *wsol:QoStype.typeName*, *wsol:qValue* and *wsol:qUnit.unitName* of the *QoSConstraint*. If it is related to a *statement*, the instance must be characterized by an expression having parameter and unit equals to *wsol:value* and *wsol:unit.unitName* of the *statement*. Finally, a new *Policy* containing the concept instances is created. The final result is equal to the one obtained for the previous two specifications.

Chapter 5

Web Service Contract Selection

Web Service discovery is a process that consists in the identification of the services, namely *eligible services*, that fulfill functional requirements given by the user. Since more than one *eligible service* is likely to be discovered, a ranking mechanism is needed in order to provide support for the automatic or semi-automatic selection of a restricted number of services (usually one) among the discovered ones. The selection consists in identifying the Web services that offer contracts including the sets of NFPs that better fulfill the NFP requirements given by the user.

The Web service contract selection can be formalized as follows: given a set of offered service contracts $SC = \{sc_1, ..., sc_n\}$ associated with the eligible services where each sc_i offers a set of NFPs, and a requested service contract R specified in terms of non-functional requirements, the *Web service contract selection* consists in defining a sorting of SC based on R.

The sorting is based on the matching between the set of NFPs offered by each sc_i and the set of NFPs specified in R. Matching between NFPs should be considered not crispy and degrees of satisfaction should be evaluated (i.e., NFP requirements can be fully, partially or not satisfied). In order to perform this evaluation several aspects must be considered:

- **different perspectives**: service providers and consumers can express their offered and required NFPs using different perspectives. As an example, in the logistic operator domain (see Chapter 2), a client company asks for a freight transportation specifying a delivery location and a logistic operator offers a freight transportation service covering a particular geographical area. This means that matching rules for the identification of comparable NFPs must be defined.

- **quantitative NFP evaluation**: proper functions for quantitative NFPs evaluation must be defined. Quantitative NFP evaluation consists in computing degrees of satisfaction between NFPs expressed with quantitative constraint operators (e.g., $=$, $\leq$, $\geq$) and numerical values. As an example, in the case study a degree of satisfaction between the request for a service with *a price less than or equal to 120 Euros* and an offer of a service with *a base price equal to 100 Euros* must be evaluated.

- **qualitative NFP evaluation**: proper functions for qualitative NFPs evaluation must be defined. Qualitative NFP evaluation consists in computing degrees of satisfaction between NFPs expressed with qualitative constraint operators (e.g., *equal, all, exist, include*) and object values. As an example, in the case study a degree of satisfaction between the request for a service with *a fire insurance or any insurance type that includes it* and an offer of a service offering a *blanket insurance* must be evaluated.

- **global offer evaluation**: a proper function for the evaluation of a global degree for each offered service contract based on degrees of satisfaction between offered and requested NFPs must be defined.

The Web service contract selection can be executed using non-semantic or semantic approaches.

Non-semantic approaches (e.g., [22, 23]) are characterized by high time efficiency but low precision due to the management of only syntactic service contract descriptions. These approaches do not support mediation between NFPs expressed using different perspectives. Moreover, they reduce the evaluation of degrees of satisfaction between qualitative NFPs to the syntactical comparison among values.

Semantic approaches (e.g., [21, 24]) utilize automated reasoning techniques on semantic service contract descriptions. These techniques are particularly suitable to mediate between different terminologies and data models considering the semantics of the terms used in the descriptions as defined by means of logical axioms and rules. Moreover, reasoning activities can be used for the evaluation of degrees of satisfaction between qualitative NFPs in order to exploit semantic relations between object values. However, logical reasoning techniques are unsuitable when dealing with non trivial numeric functions (e.g., weighted sums) which can be used for the evaluation of degrees of satisfaction between quantitative NFPs and for the global offer evaluations.

This dissertation proposes an hybrid approach to Web service contract selection that combine logic-based and algorithmic techniques. Logic-based techniques are used for mediation and qualitative NFP evaluation instead more practical algorithmic techniques are used for quantitative NFPs and global offer evaluations. The aim of the proposal is to overcome the limitations of purely semantic or non-semantic approaches using logical reasoning techniques on semantic Web service contract descriptions only when they are strictly necessary to improve the precision of the selection activity.

Moreover, the hybrid approach offers good levels of:

- **expressivity**, by supporting PCM-based WSML descriptions (see Chapter 3) of requested and offered NFPs addressing qualitative properties by mean of logical expressions on ontology values and quantitative properties by mean of expressions including ranges and inequalities;

- **generality**, by allowing semantic-based mediation between NFP descriptions based on multiple ontologies through the definition of logical axioms and rules;

- **extensibility**, by supporting the customization of functions to be used for NFP and global offer evaluations;

- **flexibility**, by allowing the selection also in case of incomplete specifications (i.e., unspecified properties or expressions including range of values in NFP offers).

This chapter proceeds as follows. Section 5.1 presents the hybrid approach as an effective and flexible solution for *the Web service contract selection problem*. The effectiveness of the hybrid approach has been tested by implementing two different prototypes of the Policy Matchmaker and Ranker (PoliMaR) tool. The two prototypes are described in Section 5.2 and Section 5.3. Experimental results are provided in Section 5.2.2 and Section 5.3.1.

5.1 An Hybrid Approach to Web Service Contract Selection

The hybrid approach supports PCM-based WSML descriptions of requested sets (i.e., *RequestedPolicies*) and offered set (i.e., *Policies*) of NFPs addressing qualitative and quantitative properties. Each *Policy* must be ranked by evaluating a degree of satisfaction between the corresponding set of offered *PolicyNfp* and the set of *Requests* specified by a user in the *RequestedPolicy*.

As stated above, the hybrid approach to Web service contract selection is based on the combination of automated reasoning and algorithmic techniques. In fact, on the one hand, reasoning is important to exploit semantic dependencies of the properties and to deal with qualitative properties ranging over object values; on the other hand, NFPs often concern properties whose values are numeric and their evaluation can be hardly handled by reasoners.

As shown in Figure 5.1, the Web service contract selection process takes as input a *RequestedPolicy* specifying a set of *Request*s and several *Policies* offering different sets of *PolicyNfp*s.

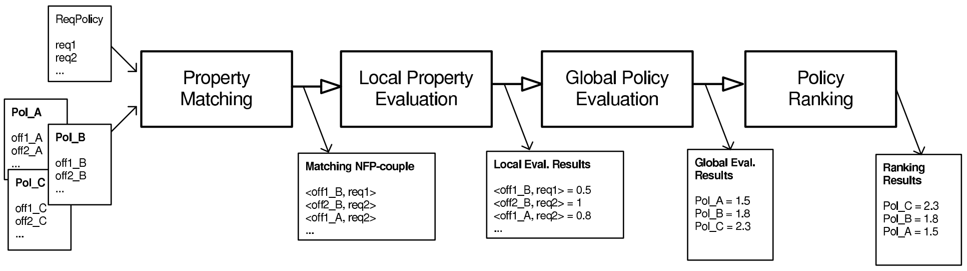


Figure 5.1: The Web service contract selection process

The process is composed of the following four phases:

- **property matching phase**: identifies the *PolicyNfp*s in the offered *Policies* that match with the *Requests* in the *RequestedPolicy*;

- **local property evaluation phase**: for each identified *Request/PolicyNfp* couple, evaluates how the offered property satisfies the requested one - results are in range [0, 1];

- **global policy evaluation phase**: for each policy, evaluates the results of the previous phase to compute a global satisfaction degree - results are values in range [0, n];

- **policy ranking phase**: sorts the *Policies* according to their global satisfaction degree.

In order to clarify the activities performed in each phase, the problem to rank *premiumPolicy* (see Listing 3.10 in Chapter 3) and *silverPolicy* (see Listing A.2 in Appendix A) against *LORequest1* (see Listing 3.11 in Chapter 3) will be used as a running example.

5.1.1 The Property Matching Phase

According to the approach based on decoupling the matching phase from the evaluation phase, the *property matching phase* has two goals:

- **Goal 1**: discover the *PolicyNfps* in the offered *Policies* that match with the *Requests* in the *RequestedPolicy*. Results are *Request/PolicyNfp* couples;
- **Goal 2**: retrieve all the data concerning these NFPs to support the other phases in the evaluation tasks.

A mediator-centric approach is used to achieve these goals and to solve semantic mismatches. In this case, the mediation is defined by logic programming rules.

Goal 1 is reached through a first set of rules that mediates among the possibly different ontologies on which *PolicyNfps* and *Requests* are based on. These *matching rules* retrieve a set of *Request/PolicyNfp* couples exploiting subclass relations (i.e., a request and an offer match if they belong to specific subclasses of *PolicyNfp*). Listing 5.1 shows an example of matching rule stating that: a *Request* defined as an instance of the concept *BasePriceRequest* (specified in an ontology with nfpo# namespace) matches with *PolicyNfps* defined as instances of *BasePrice* or *ServicePrice*.

Listing 5.1: An Example of Rule for Matching Couple Discovery

```
axiom BasePriceMatching
    definedBy
        matchCouple(?request,?nfp,baseprice) :-
            (?request memberOf nfpo#BasePriceRequest) and
            (?nfp memberOf nfpo#BasePrice) or
            (?nfp memberOf nfpo#ServicePrice)
```

The rule in Listing 5.2 is defined to reach Goal 2 retrieving the data related to each *Request/PolicyNfp* matching couple. The reasoner exploits standard mechanisms of variable binding to explore the PCM-compliant ontologies and retrieve the information for each couple. Moreover, retrieval of such data is not straightforward because non monotonic rules are exploited to put results in a kind of normal form (e.g., some quantitative properties might be defined through binary operators in some policies and ternary operators in other policies).

Listing 5.2: Rule for Request/PolicyNfp Data Retrieval

```
axiom Request2PolicyNfpMatching
    definedBy
        matchRequest2PolicyNfp(?matchType,?reqPolicy,?req,?rel,?opReq,?reqMinV,?reqMaxV,?
             reqUnit,?policy,?ws,?nfp,?opNfp,?nfpMinV,?nfpMaxV,?nfpUnit) :-
          matchCouple(?req,?nfp,?matchType) and
          (?policy [pcm#hasServiceReference hasValue ?ws] and
          ?policy [pcm#hasNfp hasValue ?nfp] and
            (?nfp [pcm#hasExpression hasValue ?x] and
              ?x [pcm#hasOperator hasValue ?opNfp] and
              ?x [pcm#hasMinParameter hasValue ?nfpMinV] and
              ?x [pcm#hasMaxParameter hasValue ?nfpMaxV] and
              ?x [pcm#hasUnit hasValue ?nfpUnit])) and
          (?reqPolicy [pcm#hasNfp hasValue ?req] and
            ?req [pcm#hasRelevance hasValue ?rel] and
            (?req [pcm#hasExpression hasValue ?y] and
              ?y [pcm#hasOperator hasValue ?opReq] and
              ?y [pcm#hasMinParameter hasValue ?reqMinV] and
              ?y [pcm#hasMaxParameter hasValue ?reqMaxV] and
              ?y [pcm#hasUnit hasValue ?reqUnit])).
```

The retrieved data for each *Request/PolicyNfp* matching couple are: (i) the ID of the matching type (i.e., *matchType*); (ii) the ID of the *RequestedPolicy* (i.e., *reqPolicy*) that includes the *Request*; (iii) the ID of the Request (i.e., *req*) and the related relevance, constraint operator, minimum value, maximum value and unit of measure (i.e., *rel, opReq, reqMinV, reqMaxV, reqUnit*); (iv) the ID of the *Policy* (i.e., *policy*) that includes the *PolicyNfp* and the Web Service (i.e., *ws*) offering it; (v) the ID of the *PolicyNfp* (i.e., *nfp*) and the related constraint operator, minimum value, maximum value and unit of measure (i.e., *opNfp, nfpMinV, nfpMaxV, nfpUnit*).

Formally, the results of the matching consist in a table with all the data necessary for the next phases. The results of the *property matching phase* applied to *premiumPolicy*, *silverPolicy* and *LORequest1* are sketched in Table 5.1.

5.1.2 The Local Property Evaluation Phase

The *local property evaluation phase* takes as input the result table of the property matching phase and evaluate, for each identified *Request/PolicyNfp* match-

Table 5.1: The *property matching phase* results

reqPolicy/policy	req/nfp	operator	MinV	MaxV	Unit	Rel.	WS
LOReqPolicy1	**reqBasePrice**	lessEqual	120	null	euro	0.8	null
premiumPolicy	**offBasePrice1**	equal	100	null	euro	-	WSOmniTransport
silverPolicy	**offServicePrice2**	equal	150	null	euro	-	WSFastTransport
LOReqPolicy1	**reqPaymentDeadline**	atLeast	15	null	days	0.4	null
premiumPolicy	**offPaymentDeadline1**	equal	45	null	days	-	WSOmniTransport
silverPolicy	**offPaymentDeadline2**	equal	60	null	days	-	WSFastTransport
LOReqPolicy1	**reqHoursToDelivery**	interval	24	48	hours	0.8	null
premiumPolicy	**offHoursToDeliveryRange1**	interval	24	48	hours	-	WSOmniTransport
silverPolicy	**offHoursToDelivery2**	equal	24	null	hours	-	WSFastTransport
LOReqPolicy1	**reqPaymentMethod**	exist	carriagePaid,carriageForward	null	null	0.6	null
premiumPolicy	**offPaymentMethod1**	all	carriagePaid	null	null	-	WSOmniTransport
silverPolicy	**offPaymentMethod2**	all	carriagePaid,carriageForward	null	null	-	WSFastTransport
LOReqPolicy1	**reqInsurance**	include	fireInsurance	null	null	0.6	null
premiumPolicy	**offInsurance1**	all	blanketInsurance	null	null	-	WSOmniTransport
silverPolicy	**offInsurance2**	all	fireInsurance	null	null	-	WSFastTransport

ing couple, how the *PolicyNfp* satisfies the *Request*. The output of this phase is a *local satisfaction degree* (*LD* for short) for each couple. A LD is expressed by a value in the range $[0..1]$, where 0 means "*no match*" and 1 means "*exact match*". The LD is computed through an evaluation function selected on the basis of the constraint operators specified in the *Request/PolicyNfp* matching couple. Links between functions and constraint operators are not fixed. To supply a flexible and extensible solution, users can change these links specifying new evaluation functions.

Different techniques must be used to evaluate matching couples regarding quantitative and qualitative NFPs. The local quantitative property evaluation requires only the usage of mathematical functions, like the ones introduced in [48]. On the contrary, for the local qualitative evaluation, the reasoner needs to be recalled to exploit inference mechanisms based on the NFP domain ontologies in use.

The Local Quantitative Property Evaluation

For each quantitative NFP matching couple, the LD is calculated by a function that takes the form $e\left(cop_r, cop_o, norm\left(v_r\right), norm\left(v_o\right)\right)$, where cop_r and cop_o are the requested and offered constraint operators; $norm\left(v_r\right)$ and $norm\left(v_o\right)$ are the requested and offered normalized values (i.e., values after a unit conversion when necessary).

Table 5.2 shows examples of functions that can be used to perform the local quantitative property evaluation. As stated above, each functions is associated with a couple of constraint operators used in the *Request* expression (*Req-C.Op.*) and in the *PolicyNfp* expression (*Off-C.Op.*). In the formulas, in case of binary constraint operators, r indicates the requested value and o the offered value. For what concern ternary operators, $minR$ and $maxR$ indicated the minimum and maximum requested values, and $minO$ and $maxO$ indicated the minimum and maximum offered values.

Considering the *Request/PolicyNfp* couples shown in Table 5.1, the local property evaluation is performed as follow: (i) formula (3) is used for the evaluation of <*reqBasePrice,offBasePrice1*> and <*reqBasePrice,offServicePrice2*>; (ii) formula (6) is used for <*reqPaymentDeadline,offPaymentDeadline1*> and <*reqPaymentDeadline,offPaymentDeadline2*>; (iii) formula (8) is used for <*reqHoursToDelivery,offHoursToDelivery2*> and (iv) formula (9) is used for <*reqHoursToDelivery,offHoursToDeliveryRange1*>. The results of the evaluation

	Req-C.Op.	Off-C.Op.	LD
(1)	$equal$	$equal$	$LD = \begin{cases} 1 & \text{if } r = o \\ 0 & \text{elsewhere} \end{cases}$
(2)	$equal$	$lessEqual$	$LD = \begin{cases} 1 & \text{if } r \leq o \\ 0 & \text{elsewhere} \end{cases}$
(3)	$lessEqual$	$equal$	$LD = \begin{cases} 1 - e^{-\frac{(r-o)^2}{r^2+1}} & \text{if } r > o \\ 0 & \text{elsewhere} \end{cases}$
(4)	$lessEqual$	$lessEqual$	$LD = \begin{cases} 1 & \text{if } r > o \\ e^{-\frac{(r-o)^2}{r^2+1}} & \text{elsewhere} \end{cases}$
(5)	$lessEqual$	$interval$	$LD = \begin{cases} 0 & \text{if } \nexists o \in [minO, maxO] \cap [-\infty, r] \\ 1 - \frac{r-minO+maxO}{maxO-minO} & \text{if } [minO, maxO] \subset [-\infty, r] \\ 1 - \frac{maxO}{maxO-minO} & \text{if } minO \in [-\infty, r] \text{ e } maxO \notin [-\infty, r] \\ 1 - \frac{r-minO}{maxO-minO} & \text{if } minO \notin [-\infty, r] \text{ e } maxO \in [-\infty, r] \\ 1 & \text{elsewhere} \end{cases}$
(6)	$atLeast$	$equal$	$LD = \begin{cases} 0 & \text{if } r > o \\ e^{-\frac{(r-o)^2}{r^3}} & \text{elsewhere} \end{cases}$
(7)	$atLeast$	$lessEqual$	$LD = \begin{cases} 0 & \text{if } r > o \\ e^{-\frac{(r-o)^2}{r^3}} & \text{elsewhere} \end{cases}$
(8)	$interval$	$equal$	$LD = \begin{cases} 1 & \text{if } o \in [minR, maxR] \\ 0 & \text{elsewhere} \end{cases}$
(9)	$interval$	$interval$	$V = \begin{cases} 0 & \text{if } \nexists o \in [minO, maxO] \cap [minR, maxR] \\ 1 - \frac{minR-minO+maxO-maxR}{maxO-minO} & \text{if } [minO, maxO] \subset [minR, maxR] \\ 1 - \frac{maxO-maxR}{maxO-minO} & \text{if } minO \in [minR, maxR] \text{ e } maxO \notin [minR, maxR] \\ 1 - \frac{minR-minO}{maxO-minO} & \text{if } minO \notin [minR, maxR] \text{ e } maxO \in [minR, maxR] \\ 1 & \text{elsewhere} \end{cases}$

Table 5.2: Examples of Local Quantitative Property Evaluation Functions

are shown in Table 5.3.

The Local Qualitative Property Evaluation

The local qualitative evaluation required the usage of the reasoner to exploit inference mechanisms regarding the *Request/PolicyNfp* couples to be evaluated. Qualitative *Request* can be expressed using the constraint operators *all*, *exist* and *include* defined in the PCM (see Chapter 3). Only the operator *all* can be used for qualitative *PolicyNfp* expressions.

The operators all and $exist$ have standard logical meaning; basic inferences based on identities need to be considered for both the operators (e.g., when a service ships to *Italy* and the request ask for a service shipping to *Italia*). Let V be the set of requested values and O the set of offered values. For *Request/PolicyNfp* couples based on the all operator, the LD is evaluated using the following function:

$$LD = \begin{cases} 1 & \text{if } V \subseteq O \\ 0 & \text{if } V \cap O = \emptyset \\ \frac{|V \cap O|}{|V|} & \text{if } V \nsubseteq O \text{ e } V \cap O \neq \emptyset \end{cases}$$

The operator *exist* means that at least one of the requested value must be available in the set of the offered values. For *Request* based on the *exist* operator, the evaluation is performed according to the following function:

$$LD = \begin{cases} 1 & \text{if } \exists v \in V \text{ tale che } v \in O \\ 0 & \text{elsewhere} \end{cases}$$

*Request*s specified through an $include$ operator need to be evaluated considering specific dependencies with the values specified in the *PolicyNfp*s. As an example, the insurance ontology (see Listing B.10 in Appendix B) defines the *fireInsurance* as a *partOf* of the *blanketInsurance*. Therefore, a *Policy* offering a *blanketInsurance* satisfies a *RequestedPolicy* asking for a *fireInsurance*. A mediator-centric approach is used to exploit these relations. As an example, the axiom in Listing 5.3 states that the *partOf* relation among insurance is to be considered as an inclusion relation.

The local evaluation function for inclusion operators expands the set O of offered values according to the transitive closure for the inclusion relations involving offered and requested values. Then, LD is calculated as for the all operator.

Listing 5.3: Rule for Inclusion Relations

```
axiom insuranceInclusion
    definedBy
      include(?X,?Y) :-
        (?X memberOf ins#Insurance) and (?Y memberOf ins#Insurance) and ins#partOf(?X,?Y)
```

The results of the local property evaluation for the *Request/PolicyNfp* couples identified in the property matching phase are shown in Table 5.3.

Request	PolicyNfp	LD
reqBasePrice	offBasePrice1	0.03
reqBasePrice	offServicePrice2	0
reqPaymentDeadline	offPaymentDeadline1	0.76
reqPaymentDeadline	offPaymentDeadline2	0.54
reqHoursToDelivery	offHoursToDeliveryRange1	1.0
reqHoursToDelivery	offHoursToDelivery2	1.0
reqPaymentMethod	offPaymentMethod1	1.0
reqPaymentMethod	offPaymentMethod2	1.0
reqInsurance	offInsurance1	1.0
reqInsurance	offInsurance2	1.0

Table 5.3: The *local property evaluation phase* results

5.1.3 The Global Policy Evaluation Phase

The *global policy evaluation phase* takes the set of LDs evaluated for each *Request/PolicyNfp* couple as input, and provides a *global satisfaction degree* (*GD* for short) as output. GD provides information about how much a *Policy* matches a *RequestedPolicy* and it is computed by taking into account the relevance associated with each *Request* in the *RequestedPolicy*. Different Multi-Criteria Decision Making (MCDM) approaches can be used for the GD evaluation. An example is the Simple Additive Weighting (SAW) technique that consists in multiplying the normalized value of each LD for the relevance value of the corresponding *Request*. The formula is defined as follows:

$$GD = \sum_{i=1}^{n} LD_i * rel_i$$

Alternatives to the SAW technique can be revised versions of TOPSIS (Technique for Order Preference by Similarity to the Ideal Solution), AHP (Analytical Hierarchy Process) and SMART (Simple Multi Attribute Rating Technique).

Observe that the proposed approach is tolerant w.r.t. the incompleteness of the NFP specifications (i.e., *Requests* whose matching *PolicyNfps* are not specified in a *Policy*). In fact, the more *Requests* in the *RequestedPolicy* match with some *PolicyNfps* for a given *Policy*, the greater the GD is; however, the evaluation does not crash when a *Request* in the *RequestedPolicy* does not match with any *PolicyNfps* of a given *Policy*.

The results of the global policy evaluation for the LDs shown in Table 5.3 through the application of the SAW technique are represented in Table 5.4.

RequestedPolicy	**Policy**	**GD**
LORequest1	premiumPolicy	2.328
LORequest1	silverPolicy	2.216

Table 5.4: The *global policy evaluation phase* results

5.1.4 The Policy Ranking Phase

The *policy ranking phase* consists in ranking the *Policies* according to their GD. Different techniques can be used to perform this activity. The simplest technique consists in ordering the *Policies* using a traditional sorting algorithm.

A different technique tackles the problem regarding the applicability condition of a *Policy*. As stated in Chapter 3, PCM-based specifications allow the definition of applicability conditions that often cannot be automatically evaluated at run-time. This is due to the fact that the evaluation of these conditions can require several data from the users (e.g., personal information) and not only his/her preferences. An applicability condition can determine the impossibility to apply a policy to a particular user. Considering this fact, *Policies* can be divided in two different sets (i.e., *Policies* with/without applicability condition) before applying the sorting algorithm. Only the set including *Policies* without applicability conditions is considered when a service must be automatically invoked and there is not the possibility to get additional data from the user.

5.2 The PoliMaR tool: First Prototype

The process of enabling Web service contract selection can be divided in two phases:

- *setup-time*: a number of *Policies* are stored into the `Ontology Repository` together with all the ontologies necessary for their evaluation. Moreover, a `Configuration File` defining configuration parameters to be used along the selection process is specified.

- *run-time*: a *RequestedPolicy* is submitted to the engine and the *property matching*, *local property evaluation*, *global policy evaluation* and *policy ranking* activities are performed. The result is a list of *Policies* ordered respect to their compliance with the *RequestedPolicy*.

The PoliMaR tool supports setup-time and run-time activities, from the storages of *Policies* by the service providers and the submission of *RequestedPolicy* by the clients, to the definition of the ranked list of *Policies*.

Two different prototypes of the PoliMaR tool have been developed in order to provide and evaluate different solutions for the implementation of the hybrid selection approach. The architecture of the first prototype of the PoliMaR tool, illustrated in Figure 5.2, is composed of independent modules that supply services through an API that gives access to:

- a `Ontology Manager`, which is in charge of receiving *Policy* descriptions and storing them into an appropriate repository;

- an `Execution Engine`, which receives the *RequestedPolicy* submitted by the client and implements the execution strategies to fulfill the selection process;

- a `Configuration Manager`, which allows the client to specified configuration parameters to be used to perform the selection.

The `Execution Engine` relies on a set of components providing for specific features that can be extended by new components without disrupting the architecture. Since the adopted interaction model prevents components to communicate each other, they act as servers that provide their services to the Execution Engine and make it the orchestrator of the selection process, which can enact different workflows to implement different selection logics.

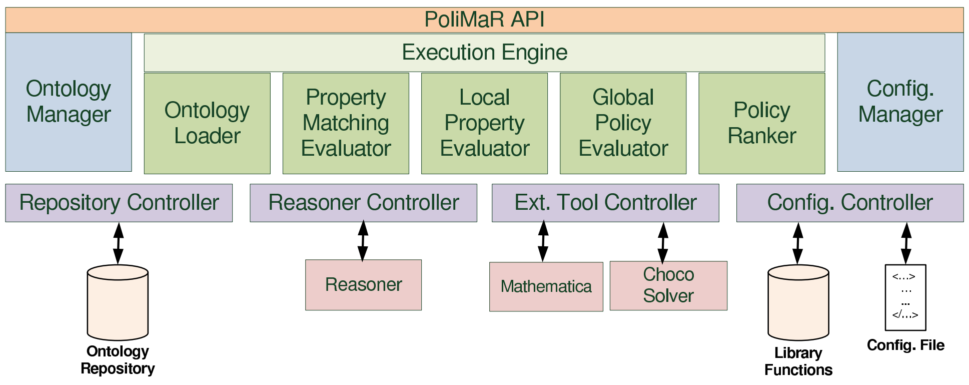


Figure 5.2: The architecture of the PoliMaR tool (first prototype)

The PoliMaR components rely on common services (namely, `Repository Controller`, `Reasoner Controller`, `External Tool Controller` and `Configuration Controller`). These components have the primary task of decoupling PoliMaR from any specific platform or technology. As an example, the `Reasoner Controller` aims at making the upper components independent of the actual underlying reasoner and its syntax. Since the WSML syntax is adopted, the current implementation of the Reasoner Controller exploits the WSML2Reasoner framework[1], which is a tool that combines various validation, normalization, and transformation functionalities essential to the translation of ontology descriptions from WSML to the appropriate syntax of several underlying reasoning engines. This way, the choice of the underlying reasoner can be easily changed at any time.

5.2.1 Core Components

The first prototype of the PoliMaR tool has been implemented using Java JDK 1.6.0 update 11 for Linux 64 bit. All the ontologies are represented in the WSML language. The Wsml2Reasoner API (v0.6.1) are used to communicate with the reasoner. This prototype is part of the GLUE2 discovery engine [51] available at http://glue2.sourceforge.net.

In this section, the main characteristics of the PoliMaR core components, data and external tools are described. The PoliMaR core components are the following:

- `Ontology Manager`: is responsible for getting WSML files, parsing them

[1]http://tools.deri.org/wsml2reasoner/

in order to retrieved relevant information (e.g, URI of the *Policy*, reference to the service offering the *Policy*), and storing them into the `Ontology Repository` through the `Repository Controller`.

- `Configuration Manager`: is used for getting new evaluation functions and configuration parameters (e.g., associations between couples of constraint operators and the functions to be used for their evaluation) from the users. Through the `Configuration Controller`, new functions are stored into the `Library Functions` and configuration parameters are saved into the `Configuration File`.

- `Ontology Loader`: is in charge of loading all the knowledge necessary for the *property matching* and *local qualitative property evaluation* phases into the `Reasoner` through the `Reasoner Controller`. The knowledge consists in all the ontologies stored in the `Ontology Repository` and the ontology specifying the *RequestedPolicy*.

- `Property Matching Evaluator`: implements the process necessary to perform the *property matching* phase described in Section 5.1.1. Through the `Reasoner Controller`, this component submits the head of the rule in Listing 5.2 to the `Reasoner` and receives results like the one in Table 5.1.

- `Local Property Evaluator`: implements the process necessary to perform the *local property evaluation* phase described in Section 5.1.2. From the result table produced by the `Property Matching Evaluator`, this component extracts the constraint operators used to express each *Request/PolicyNfp* couple. Each extracted *Request-C.Op./PolicyNfp-C.Op.* couple is used to retrieve, through the `Configuration Controller`, the function to be used for the LD evaluation. The associations between *Request-C.Op./PolicyNfp-C.Op.* couples and evaluation functions are taken from the `Configuration File`, instead evaluation functions are retrieved from the `Library Functions`. The evaluation proceeds loading and executing the retrieved function on the values of the analyzed *Request/PolicyNfp* couple. As discussed in Section 5.1.2, functions used to evaluate quantitative couples perform compute the LD between numerical values. These computations can require the usage of external tools (e.g., Mathematica, Choco Solver) accessible through the `External Tool Controller`. Viceversa, functions for qualitative couple submits queries to the `Reasoner` through the `Reasoner Controller` to retrieve additional knowledge about relations (e.g., equality, inclusion) between *Request/PolicyNfp* values. The result produced by this component is a LD for each *Request/PolicyNfp* couple (see Table 5.3).

- `Global Policy Evaluator`: implements the process necessary to perform the *global policy evaluation* phase described in Section 5.1.3. This component retrieves from the `Library Functions`, through the `Configuration Controller` and the `Configuration File`, the function to be used for the GD evaluations. The function is loaded and executed on the LD values computed by the `Local Property Evaluator`. Complex functions can require external tools for the computation. The usage of these tools is mediated by the `External Tool Controller`. The result produced by this component is a GD for each *RequestedPolicy/Policy* couple (see Table 5.4).

- `Policy Ranker`: implements the process necessary to perform the *policy ranking* phase described in Section 5.1.4. Also this component retrieves from the `Library Functions`, through the `Configuration Controller` and the `Configuration File`, the function to be used to perform the ranking. The function is loaded and executed on the GD values produced by the `Global Policy Evaluator`.

The PoliMaR tool uses the following data and external tools:

- `Ontology Repository`: includes several WSML ontologies among which: (i) the `PCM ontology`; (ii) the `SeCO Reference Ontology` (see Chapter 4) providing PCM-based descriptions of heterogenous NFPs such as QoS, business, context and license terms; (iii) the `Policy Repository Ontology` including 500 PCM compliant *Policy* instances; (iv) the `Rule Ontology` specifying all the axioms/rules needed for the selection process; (v) several `Domain Ontologies`, such as the *Insurance Ontology*, extending the SeCO Reference Ontology with NFPs related to a particular domain. Complete specifications of these ontologies are in Appendix A, B and C.

- `Reasoner`: is used to extract knowledge from the available ontologies. The current version of the PoliMaR uses KAON2 (v2007-06-11)[2].

[2]http://kaon2.semanticweb.org/

- `External Tools` (e.g., Mathematica[3] and Choco Solver[4]): are used when sophisticated evaluations are needed. As described above, they can be used by the `Local Property Evaluator` and the `Global Policy Evaluator`.

- `Library Functions`: are the functions to be used for *local property evaluation*, *global policy evaluation* and *policy ranking*. Currently, they are defined as Java classes.

- `Configuration File`: is an XML file containing configuration parameters defined by the PoliMaR user. These parameters are: (i) the association between constraint operators and evaluation functions; (ii) the function to be used by the `Global Policy Evaluator`; (iii) the function to be used by the `Policy Ranker`.

5.2.2 Experimental Results

The first prototype of the PoliMaR has been tested to evaluate the scalability and the efficiency of the matching and evaluation components. The evaluation activity has been performed using an Intel Core2 Q6700 2.66 Ghz with 2GB RAM and Linux kernel 2.6.27 64 bits. Due to the lack of large and accessible sets of NFP descriptions to derive PCM-based descriptions, the experiment has been carried out starting from a set of randomly generated descriptions that consists of about 500 policies. The generated test set is a combination of the properties discussed in Section 2.2.1 to form policies that are described according to the SeCO Reference Ontology; constraint operators and parameters are selected randomly according to the ranges specified in the ontology.

The *RequestedPolicy LORequest1* shown in Listing 3.11 has been used as testbed. It is composed of three quantitative requests $r\#1$ (i.e., reqBasePrice), $r\#2$ (i.e., reqPaymentDeadline), and $r\#3$ (i.e., reqHoursToDelivery); and two qualitative requests $r\#4$ (i.e., reqPaymentMethod), and $r\#5$ (i.e., reqInsurance). Observe that $r\#1$, $r\#2$ and $r\#3$ are based on three different constraint operators, namely *lessEqual*, *atLeast* and *Interval* that require different functions for the evaluation. Moreover, observe that $r\#4$ and $r\#5$ are based on two different constraint operators, namely *exist* and *include*, that require different reasoning tasks for the evaluation. The performed tests were:

[3]http://wolfram.com/products/mathematica/index.html
[4]http://choco.emn.fr/

- **TEST 1**: Measurement of the overall execution time in the cases of single and multiple file storage;
- **TEST 2**: Analysis of the execution-time distribution between reasoning and algorithmic computation for single file storage;
- **TEST 3**: Analysis of the execution-time distribution among the main selection phases (matching, local and global evaluation) for single file storage;
- **TEST 4**: Measurement and comparison of the overall execution time with increasing complexity in the requested policy for single file storage.

The first part of the test activity concerned the scalability of the loading process. Single and multiple file storage are considered. In the former, all *Policies* are collected into a single WSML file containing one large ontology (i.e., the `Policy Repository`); in the latter, each `Policy` is defined in a different ontology and stored in a different WSML file.

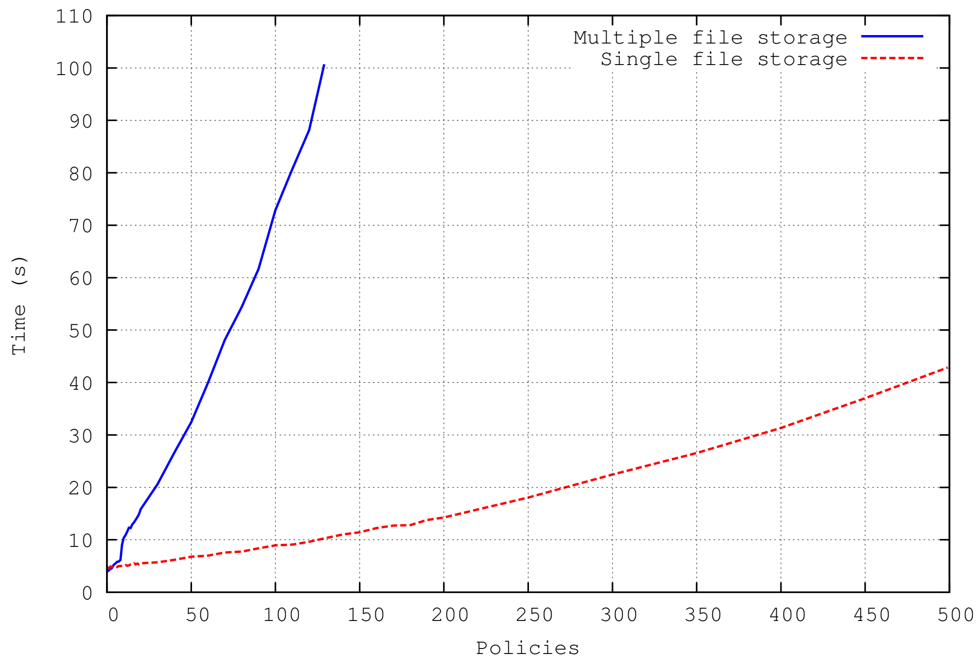


Figure 5.3: Test 1 - Results

The results of TEST 1 are shown in Figure 5.3. TEST 1 highlights that: (i) the multiple file approach is efficient only for small numbers of policies.

Moreover, the KAON2 reasoner was able to manage at most 136 WSML files containing a policy each; (ii) the required time increases exponentially for the multiple-file approach and polynomially for the single-file approach; (iii) there is an amount of time (approximately 5 seconds) that is independent of the input. It represents the time required to invoke the reasoner through the WSML2Reasoner API. The conclusion that can be driven from this first test is that semantic tools available today make the single file approach compulsory. Ongoing research on large scale distributed reasoning might overcome this limit in the future.

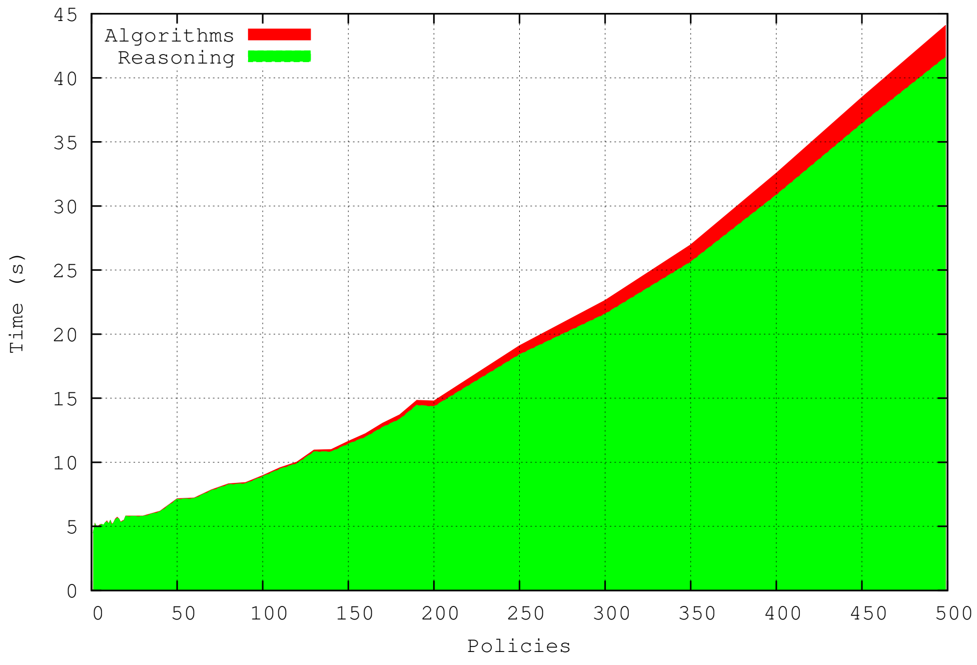


Figure 5.4: Test 2 - Results

TEST 2 aims to analyze the execution-time of the hybrid approach used by the PoliMaR. The results shown in Figure 5.4 highlights that the bottleneck for the evaluation is represented by the reasoner: the time required for the evaluation of quantitative NFPs and the global evaluation phase is very short.

TEST 3 is used to evaluate the execution-time distribution among the main ranking process phases in order to identify the bottleneck. The results in Figure 5.5 highlights that: (i) the time required for the execution of the global policy evaluation phase does not influence significantly the evaluation time; (ii) the time used for the local property evaluation phase is twice as long as the time for the property matching phase. These results are strictly related to the one

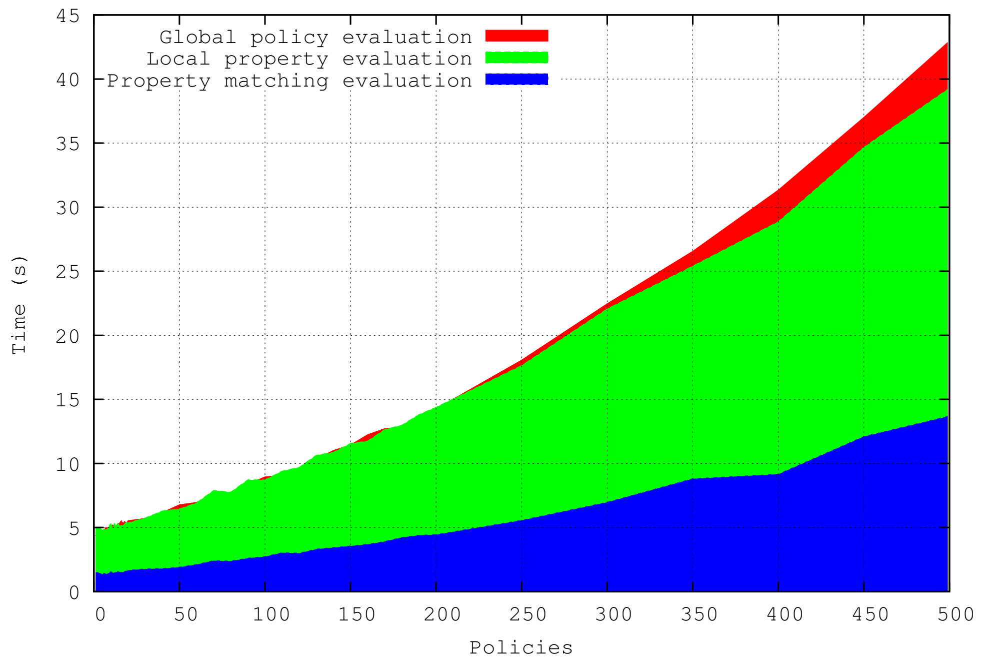


Figure 5.5: Test 3 - Results

obtained for TEST 2: the phases that influence significantly the execution-time are the ones that require the usage of the reasoner.

TEST 4 has been executed considering six combinations of the single *Requests* forming the *RequestedPolicy* with increasing degrees of complexity. The first three *RequestedPolicies* were composed of quantitative requests only: $RP1$ was composed of $r\#1$ (written $RP1 = r\#1$); $RP2 = RP1 + r\#2$; $RP3 = RP2 + r\#3$. The next three requested policies considered also qualitative requests: $RP4 = RP3 + r\#4$ (the *all* constraint operator is used); $RP5 = RP3 + r\#5$ (the *include* constraint operator is used); $RP6 = RP3 + r\#4 + r\#5$ (both *all* and *include* are used).

The results of TEST 4 (Figure5.6) highlight that: (i) the number of quantitative NFP constraints marginally affects the evaluation time (RP1, RP2 and RP3 show similar evaluation times); (ii) the evaluation of a qualitative NFP expressed with the *all* operator requires more time than one expressed with the *include* operator. Considering a `Policy Repository` with 500 policies, the introduction of an *all* constraint determines an increment of 12 seconds, instead the inclusion of an *include* constraint determines an increment of 4 seconds.

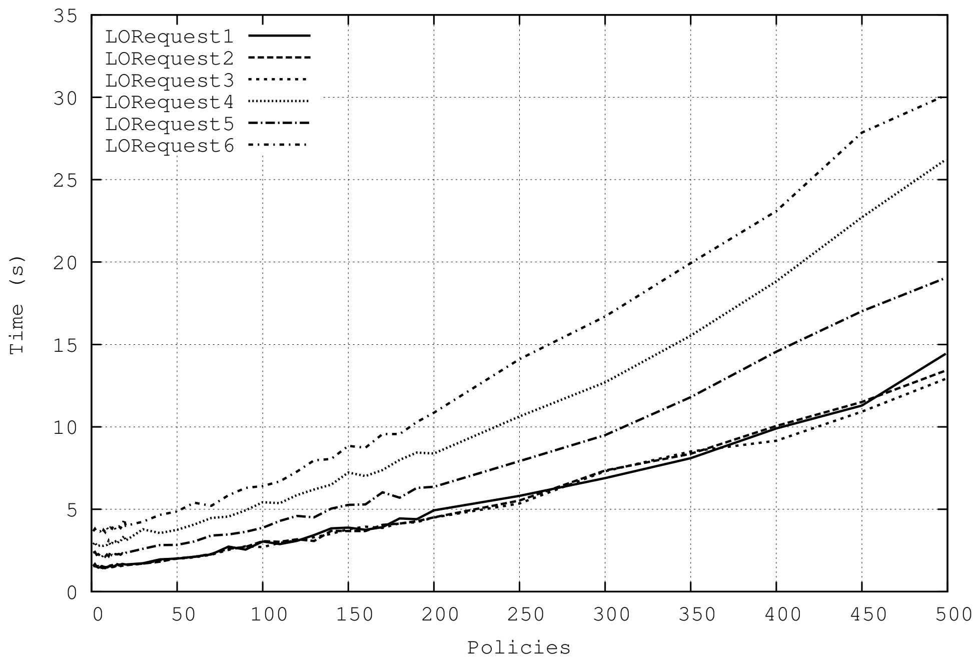


Figure 5.6: Test 4 - Results

To summarize, the main results of TEST 1-4 highlights that:

- The semantic tools available today make the single file storage approach compulsory. The soundness of the decision to use a single-file `Policy Repository`, as described in Section 5.2.1, is confirmed.
- The performance are low (i.e., 43 seconds are required to rank 500 *Policies*) and allow to use the PoliMaR only in context in which the time-efficiency is not a hard requirement. The bottlenecks are:
  - the reasoning activities required for the property matching and for the local evaluation of qualitative *Request/PolicyNfp* couple;
  - the amount of time (approximately 5 seconds), independent of the input, that is required to invoke the reasoner through the WSML2Reasoner API.

5.3 The PoliMaR tool: Second Prototype

In this section, a second prototype for the PoliMaR tool is proposed. The main objective is to increase the performance of the tool, overcoming the limitations highlighted along the evaluation activity on the first prototype. In order to preserve the hybrid approach described in Section 5.1, the only possibility to increase the performance consists in defining a new implementation strategy based on a different usage of the reasoning activities.

The proposed strategy consists in the usage of *caching* techniques in order to extract and store in advance from the available ontologies all the knowledge needed for the selection process. This strategy requires to change the process of enabling service selection described in Section 5.2, and in particular to modify the activities to be performed at *setup-time* and *run-time* as follows:

- *setup-time*: in addition to the storage of the ontologies into the `Ontology Repository` and the setting of the `Configuration File`, different types of *caches* must be created in order to make available in practical data structures all the knowledge needed for the run-time evaluation. The caches are created through the `Reasoner` that extracts relevant information from the `Ontology Repository`.

- *run-time*: a *RequestedPolicy* is submitted to the engine and the *property matching*, *local property evaluation*, *global policy evaluation* and *policy ranking* activities are performed using the knowledge stored in the `caches`. The usage of the `Reasoner` is omitted.

In order to support both *property matching* and *local property evaluation*, two different types of *caches* must be created:

- `Matching Cache`: includes, for each *Request* concept (i.e., ontological concept defined as subConceptOf pcm#Request), all the instances of *PolicyNfps* defined in the `Policy Repository` that satisfy the associated matching rule (see Listing 5.1 for an example). Moreover, it includes the reference to all the *Policies* that offer the identified *PolicyNfps*. The example of `Matching Cache` shown in Table 5.5 allows to identify all the *PolicyNfps* to be evaluated when an instance of the *RequestedInsurance* or *RequestedPaymentMethod* concepts occur in a *RequestedPolicy* submitted to the PoliMaR by an user. Observe that, in order to improve the reuse, the possibility to have the same *PolicyNfp* offered by more *Policies* is addressed. This cache enables the *property matching*, omitting the usage of the `Reasoner` at run-time.

- `Relation Cache`: includes, relations (e.g., *inclusion*, *equality*) between concept instances defined in the ontologies available in the `Ontology`

Request Concept	**PolicyNfp**	**Policy**
requestedInsurance	*offInsurance1*	*premiumPolicy*
	offInsurance2	*silverPolicy*
requestedPaymentMethod	*offPaymentMethod3*	*goldPolicy*
		seniorPolicy
	offPaymentMethod1	*premiumPolicy*
		freedomPolicy
	offPaymentMethod2	*silverPolicy*
...	...	...

Table 5.5: Example of data included in the *matching cache*

`Repository`. This cache provides the knowledge required to enable the *local qualitative property evaluation* where inference mechanisms based on the ontologies in use must be exploited. The example of `Relation Cache` shown in Table 5.6 refers to the *inclusion* relations among instances defined in the *Insurance Ontology*. The cache is generated submitting to the `Reasoner` the following query: $include(?x, ?y)$.

x	y
actOfVandalismInsurance	*fireInsurance*
	fireAndTheftInsurance
	blankedInsurance
explosionInsurance	*fireInsurance*
	fireAndTheftInsurance
	blankedInsurance
atmosphericAgentInsurance	*fireInsurance*
	fireAndTheftInsurance
	blankedInsurance
fireInsurance	*fireAndTheftInsurance*
	blankedInsurance
theftInsurance	*fireAndTheftInsurance*
	blankedInsurance
damageInsurance	*blankedInsurance*
fireAndTheftInsurance	*blankedInsurance*
lossInsurance	*blankedInsurance*

Table 5.6: Example of data included in the *relation cache*

As for the first prototype, the second prototype of PoliMaR tool has been implemented using Java JDK 1.6.0 update 11 for Linux 64 bit. All the ontologies are represented in the WSML language and the Wsml2Reasoner API (v0.6.1) are used to communicate with the reasoner. New implementation choices are the usage of WSML4J API[5] for the parsing of WSML files and the usage of hash table for the storage of the caches.

As shown in Figure 5.7, little changes to the previously defined architecture must be introduced to address the new proposed strategy. The usage of the `Matching Cache` and the `Relation Cache` require the introduction of two new components:

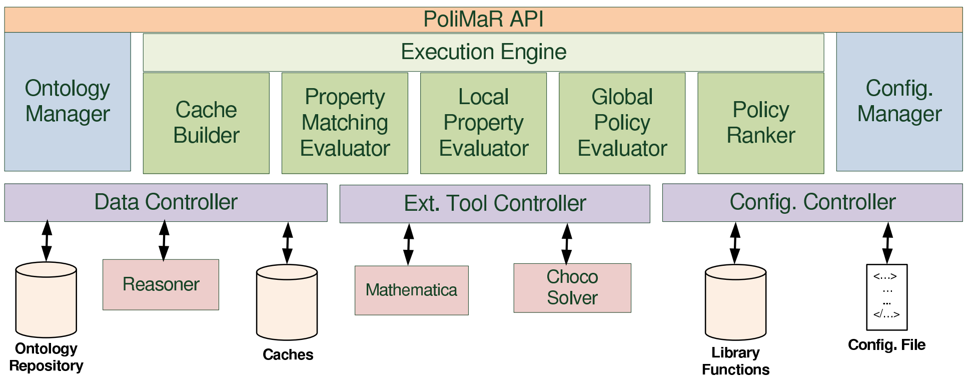


Figure 5.7: The architecture of the PoliMaR tool (second prototype)

- `Cache Builder`: substitutes the `Ontology Loader` and implements all the functionalities required for the management of the caches. It is in charge of creating the `Caches` at setup-time and updating them at run-time when new ontologies are stored in the `Policy Repository` by the `Ontology Manager`. The `Caches` are built by this component loading all the ontologies available in the `Ontology Repository` into the `Reasoner` and submitting to it proper queries (see Listings 5.1 and 5.3 for examples). This process must be repeated each time new knowledge is stored into the `Ontology Repository`. If this happens at run-time, in order to preserve data consistency, a proper concurrency technique is applied.

- `Data Controller`: substitutes the previous defined `Repository Controller` and `Reasoner Controller` and aims at making the upper

[5]http://sourceforge.net/projects/wsdl4j/

components independent of the actual underlying `Reasoner` and of the technology used to manage the `Ontology Repository` and the `Caches`.

The introduction of caching techniques requires modifications to the implementations of the following components:

- `Property Matching Evaluator`: receives a WSML file defining a *RequestedPolicy* and analyzed it through WSML4J API in order to identify *Request* instances. For each discovered instance, this component identifies the related *Request* class. This information is used to access the `Matching Cache` through the `Data Controller` and to retrieve all the *Request/PolicyNfp* couples to be evaluated and the associated *Policies*. The identified *Policies* are retrieved from the `Ontology Repository` and analyzed through WSML4J API in order to extract all the data necessary for the remaining phases of the ranking process. As an example, considering the *RequestedPolicy* LORequest1 shown in the Listing 3.11 and the `Matching Cache` in Table 5.5, the `Property Matching Evaluator` identifies the presence of the *reqInsurance* that is defined as an instance of the Request class *InsuranceRequest*. Through this information, the component retrieves from the `Matching Cache` the following *Request/PolicyNfp* couples to be evaluated: <*reqInsurance,offInsurance1*> and <*reqInsurance,offInsurance2*>. Moreover, it retrieves the information that *offInsurance1* and *offInsurance2* are offered by *premiumPolicy* and *silverPolicy* respectively. The process executed by the `Property Matching Evaluator` ends with the retrieval of *premiumPolicy* and *silverPolicy* from the `Ontology Repository` and with the extraction from them of the information shown in Table 5.7

- `Local Property Evaluator`: executes the same process described for the first prototype except for the qualitative *Request/PolicyNfp* couples evaluation. In this case, the additional knowledge about relations (e.g., equality, inclusion) between *Request/PolicyNfp* values is retrieved from the `Relation Cache` through the `Data Controller`. Coming back to the example described above, the `Relation Cache` allows the `Local Property Evaluator` to know that *blanketInsurance* offered by *premiumPolicy* satisfies *reqInsurance* since a relation of inclusion between *blanketInsurance* and *fireInsurance* is specified in the `Relation Cache` (see Table 5.5).

reqPolicy/policy	req/nfp	operator	value	Rel.	WS
LOReqPolicy1	**reqInsurance**	include	fireInsurance	0.6	null
premiumPolicy	**offInsurance1**	all	blanketInsurance	-	WSOmniTransport
silverPolicy	**offInsurance2**	all	fireInsurance	-	WSFastTransport

Table 5.7: Example of output of the `Property Matching Evaluator`

5.3.1 Experimental Results

The second prototype of the PoliMaR has been tested to evaluate the scalability and the efficiency of the matching and evaluation components. In particular, the evaluation has been performed to verify if the usage of caching techniques increases the performance of the tool, overcoming the limitations highlighted along the evaluation activity on the first prototype.

Also in this case, the evaluation activity has been performed using an Intel Core2 Q6700 2.66 Ghz with 2GB RAM and Linux kernel 2.6.27 64 bits. The experiment has been carried out on the same set of 500 policies used for the test activities of the first prototype and using the same *RequestedPolicy* (shown in Listing 3.11) as testbed. The performed tests were:

- **TEST 5**: Comparison between the overall execution-time of the two prototypes;
- **TEST 6**: Comparison between the execution-time for the *property matching evaluation* of the first prototype and the execution-time for the `Matching Cache` creation;
- **TEST 7**: Comparison between the execution-time for the *local qualitative property evaluation* of the first prototype and the execution-time for the `Relation Cache` creations.
- **TEST 8**: Analysis of the execution-time for the creation of `Matching Cache` and `Relation Caches`;

TEST 5 aims at verifying the increase of performance due to the usage of caching techniques. The overall execution-time of the first prototype is compared with the execution-time of the second prototype (i.e., the sum of the time required for the cache creation and for the selection process execution).

The results of TEST 5 are shown in Figure 5.8. TEST 5 highlights that: (i) the time required for the run-time execution of the selection process on 500 policies decreases from about 43 seconds to about 20 milliseconds. This is

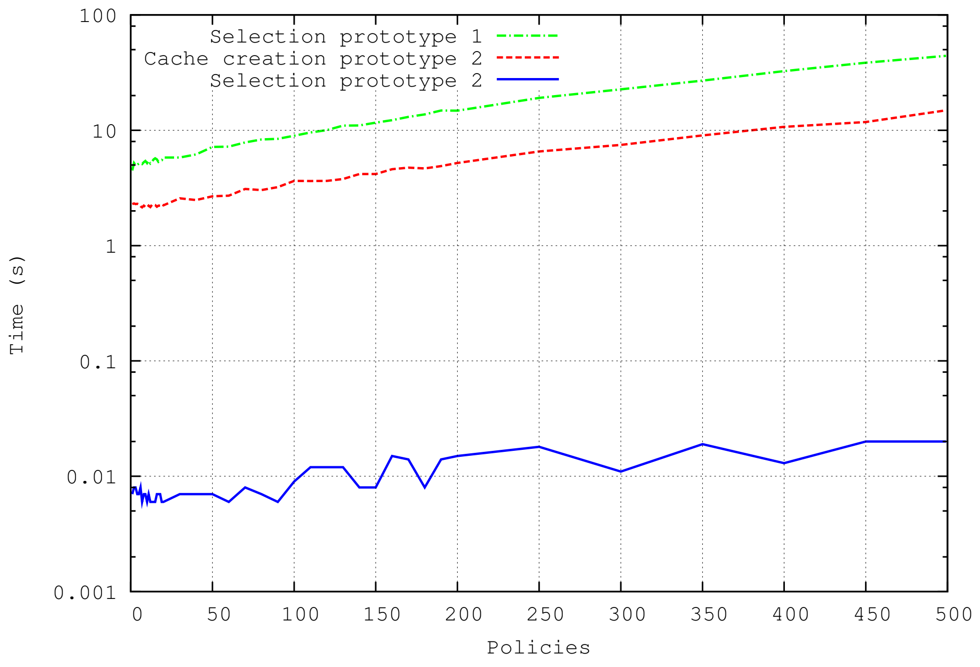


Figure 5.8: Test 5 - Results

due to the fact that usage of the `Reasoner` at run-time is omitted; (ii) the time required for the cache creation (about 15 seconds) is lower than the execution-time (about 40 seconds) for reasoning activities of the first prototype.

As stated in Section 5.2.2, the technique used by the first prototype allows the PoliMaR to be used only in context in which the time-efficiency is not a hard requirement. The former result of TEST 5 highlights that the usage of caching techniques allows the PoliMaR to be used also as a Web application. The selection process execution-time of the PoliMaR is comparable to the time required by traditional discovery/ranking engine available on the Internet.

The latter result of TEST 5 needs further analysis. TEST 6 and TEST 7 are introduced to understand what determines the decrease of execution-time.

TEST 6 is used to compare the time required by the `Property Matching Evaluator` in the first prototype to define *Request/PolicyNfp* couples to be evaluated with the time required for the `Matching Cache` creation. Both the activities are defined submitting a query to the `Reasoner`.

The results of TEST 6 are shown in Figure 5.9 and highlight that the two execution-times are similar. This means that the technique based on the usage of *Request* class used by the second prototype does not change significantly the time required for the definition of *Request/PolicyNfp* couples.

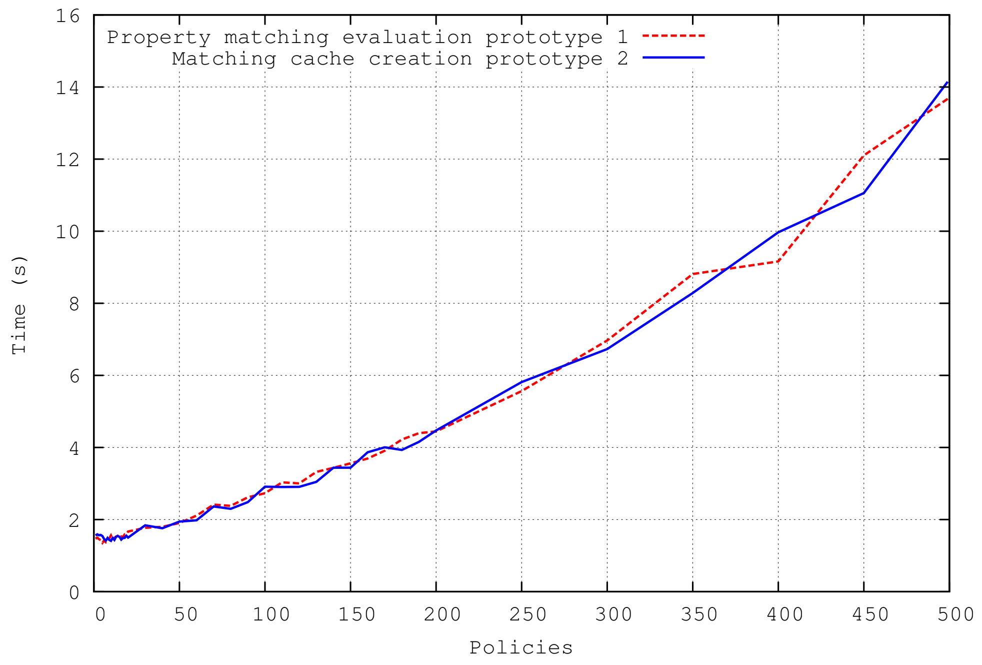


Figure 5.9: Test 6 - Results

TEST 7 consists in comparing the execution-time of the two prototype for extracting from the ontologies the information needed for the *local qualitative property evaluation*. The results of TEST 7 are shown in Figure 5.10. Different techniques are used by the two prototypes. As state above, the second prototype stores these information in the `Relation Caches` that is created submitting one query to the `Reasoner` for each considered semantic relation. In this evaluation activity, two semantic relations are considered (i.e., *equality* and *inclusion*), so two interactions with the `Reasoner` are needed. For what regards the first prototype, a query is submitted to the `Reasoner` for each value defined in the qualitative *Requests* included in the *RequestedPolicy*. In this evaluation activity, two qualitative *Requests* are considered with three specified values (i.e., *nfpo#carriagePaid*, *nfpo#carriageForward* and *ins#FireInsurance*). So, the following three queries must be submitted to the `Reasoner` (i.e., $equal(nfpo\#carriagePaid, ?y)$, $equal(nfpo\#carriageForward, ?y)$ and $include(ins\#FireInsurance, ?y)$). TEST 7 shows that the time required by the second prototype for the creation of the `Relation Caches` submitting two queries to the Reasoner is approximately 2 seconds, instead the time required by the first prototype for the *local qualitative property evaluation* on 500 *Policies* submitting three queries to the reasoner is approximately 27 seconds. This

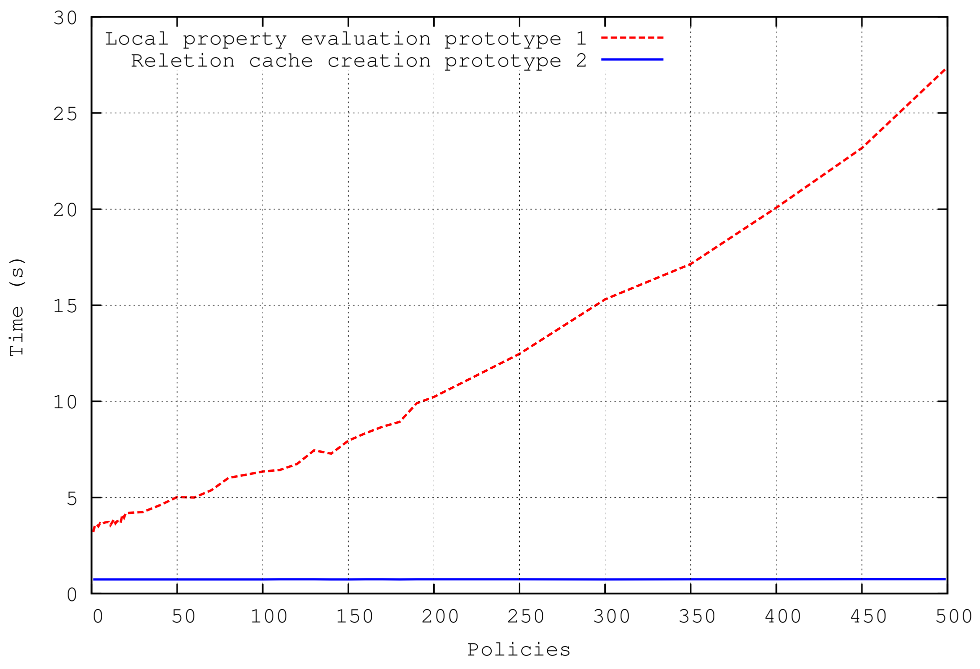


Figure 5.10: Test 7 - Results

great difference is not only determined by the different number of submitted queries, but also by the different knowledge loaded into the `Reasoner`. The first prototype through the `Ontology Loader` loads all the ontologies available in the `Ontology Repository` into the `Reasoner` at the same time. This means that the two queries for the *local qualitative property evaluation* are executed on a wide knowledge including the `Policy Repository`. Viceversa, the second prototype through the `Cache Builder` loads into the `Reasoner` only the knowledge needed for the `Relation Cache` creation (i.e., `Domain Ontologies`). Since loading/unloading knowledge into the `Reasoner` requires at least 5 seconds (see TEST 1), the strategy to repeat this activity several times is applicable only by the second prototype that interacts with the `Reasoner` at set-up time.

TEST 8 aims at comparing the execution-time for the creation of the `Matching Cache` and the two `Relation Caches` (namely, `Inclusion Cache` and `Equality Cache`). The results of TEST 8 are shown in Figure 5.11. TEST 8 highlights that: (i) the time required for the creation of the `Matching Cache` increases polynomially with respect to the number of considered *Policies*; (ii) the execution-time for the creation of the `Inclusion Cache` and the `Equality Cache` is independent of the number of considered *Policies* since they are built

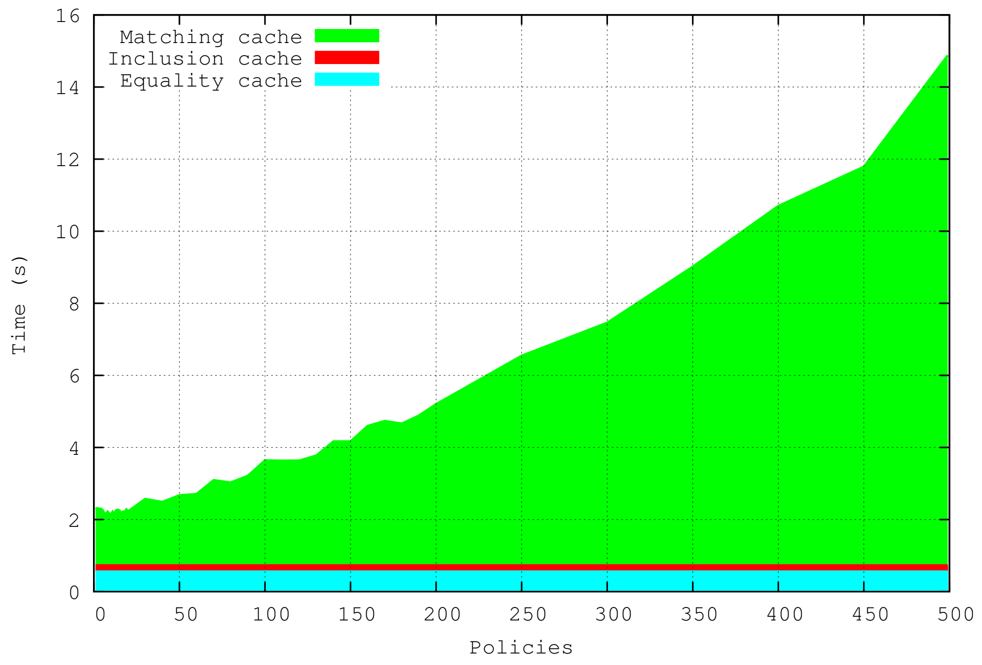


Figure 5.11: Test 8 - Results

only on the domain ontologies (i.e., the SeCO Reference Ontology and the Insurance Ontology) available in the `Ontology Repository`.

To summarize, the results of TEST 5-8 highlights that:

- the introduction of caching techniques to omit the usage of the `Reasoner` at run-time decreases the selection process execution-time. This allows the PoliMaR to be used also as a Web application since its execution time is comparable to the time required by traditional discovery/ranking engine available on the Internet.

- the technique based on the usage of *Request* class does not change significantly the time required for the definition of *Request/PolicyNfp* couples.

- the possibility to load/unload different knowledge into the `Reasoner` decreases the execution-time for the extracting from the ontologies the information needed for the *local qualitative property evaluation*.

5.4 Related Works

The analysis of the state of the art regarding NFP-based Web service selection approaches can be performed starting from the work described in [65] where a classification of approaches and a survey of the state of the art has been proposed. Three dimensions are used to perform the classification:

- *policy-based vs reputation-based*: distinguishes between approaches where NFPs are specified in policies and approaches that rely on reports from other users (i.e., reputation).

- *UDDI-extension vs Semantic Web Services*: separates approaches that propose UDDI extensions from approaches that use semantic web technology to allow NFP specifications.

- *graphic preference modeling vs ontology-based preference modeling*: distinguishes approaches based on how they capture user preferences. One set of approaches uses graphical specification of user preferences in terms of network graphs. Other approaches are textual, heavily relying on ontologies.

Using these three dimensions, the hybrid approach proposed in this dissertation can be classified as a *policy-based* solution for NFP descriptions of *Semantic Web Services* that allows *ontology-based preference modeling*.

The comparison with other approaches is carried out selecting the most relevant approaches available in the literature [21–24, 38–40] that share with our approach at least one dimension.

An hybrid architecture for Semantic Web service discovery and selection based on DL reasoners for functional discovery and Constraint Programming techniques for QoS-based selection is proposed in [38]. The architecture is independent from the language used for SWS descriptions. The QoS-based selection is limited to simple QoS and formulas for the evaluation of qualitative properties are not specified.

An extension of WS-Policy with ontological concepts to enable QoS-based semantic policy matching is presented in [39]. The extension consists in the specification of a set of elements enabling definition, parsing and reasoning on expressions related to policy assertions. However, this work considers only simple expressions: (i) only quantitative QoS characterized by a single value expression are considered. The management of quantitative properties characterized by a range value expression is not addressed; (ii) qualitative NFP are not considered both in the modeling and in the matching activities. (iii) formulas for NFP evaluation are not presented. The result of the compatibility evaluation among two policy alternatives is a boolean value. The computation of degrees of matching is not addressed.

Approaches for the WS selection based on the normalization of QoS values are described in [21, 40]. An extension of the WSMO model in order to allow a richer description of NFPs characteristics (e.g., *metricName*, *valueType*, *MeasurementUnit*) is presented in [21]. Concerning service selection and ranking, an algorithm is proposed to evaluate the matching between requested and offered QoSs. The algorithm consists in normalizing QoS values in the range [0..1], scaling the value ranges with the maximum and minimum values of each quality metric. A weighted sum is then used to evaluate the global matching degree for each provider. The limit of this approach is that it is applicable only for properties characterized by ordered scale values. Moreover, it is assumed that each property is characterized by a fixed tendency (e.g., the cost must always be minimized) limiting the freedom of the requestor in defining his requirements. The experiments performed to evaluate the approach are based on a limited set (4) of services characterized by a limited set (7) of QoSs.

A QoS computation model for web service selection based on a two-steps normalization of QoS values is proposed in [40]. Limits are the same pointed out for [21].

A NFP-based service selection approach that modifies the Logic Scoring Preference (LSP) method with Ordered Weighted Averaging (OWA) operators is proposed in [22]. A prototype to demonstrate the service selection approach is described. Even if this approach also considers the evaluation of qualitative properties, it does not consider the semantic relations among property values (e.g., *include* relation). Moreover, the prototype has been tested on a limited number of services. Information about performance and scalability are not provided.

A framework for WS selection that combines declarative logic-based matching rules with optimization methods is described in [23]. Test activities evaluates that service selection can be done in about 17 seconds on a set of 2000 policies. The proposed selection process is simplified as follows: (i) only the *equal* operator is allowed in the definition of Requests limiting the use of the reasoner for qualitative NFP evaluation; (ii) the selection process is modelled as the problem to maximize the difference between *prices* (associated with each Policy using a *pricing function*) and *score* (associated with each RequestedPolicy and stating the maximum price for which the costumer is willing to carry out the trade).

A WSMO-based hybrid solution to WS ranking based on the usage of axioms for requested and offered NFPs is defined in [24]. The exploitation of axioms supports complex and conditioned NFP definition (e.g., if the client is older than 60 or younger than 10 years old the invocation price is lower than 10 euro) but forces the user to specify all the necessary information (e.g., age) in advance.

In order to provide a more complete and schematic comparison with these approaches the following features are used:

- *expressive NFP descriptions*: the possibility to perform the selection/ranking process on NFP descriptions characterized by qualitative and quantitative values, different constraint operators, applicability conditions and relevance values;
- *semantic-based mediation*: the possibility to mediate among the different perspectives that can be used by providers and requestors to define their offers and requests.
- *parametric NFP evaluation*: the possibility to allow the specification of parameters (e.g., evaluation functions) for the NFP evaluation.
- *tolerance to unspecified NFP*: the possibility to perform the selection/ranking process also with unspecified/missing values.
- *experimental results*: certified evaluation of the approach on a significant number of NFP descriptions.

Table 5.8 reports the results of the comparison (*yes/no* is used to show whether the approach achieves the requirement, and *low, average, high* to indicate at what level the approach reaches the requirement). The table shows that, among the considered approaches, only [23] and [24] provide comparable results.

Table 5.8: Comparison of NFP-based Web service ranking approaches

	Expr. Desc.	**Mediation**	**Param. Eval.**	**Unspec. NFP**	**Experiment**
Garcia et al. 2007 [38]	low	no	no	no	no
Chaari et al. 2008 [39]	low	no	no	no	no
Liu et al. 2004 [40]	low	no	no	yes	low
Wang et al. 2006 [21]	low	no	no	yes	low
Yu et al. 2008 [22]	low	no	no	yes	low
Lamparter et al. 2007 [23]	low	yes	no	yes	high
Garcia et al. 2008 [24]	high	yes	yes	no	low
Hybrid Approach	**high**	**yes**	**yes**	**high**	**high**

Only [23] presents an evaluation activity executed on a large number of policies. Test activities demonstrates that the semantic service selection described in [23] is more efficient. However, that approach is based on simpler NFP descriptions. As stated above, only the *equal* operator is allowed in the definition of qualitative NFP and quantitative NFPs defined as range of values are not considered. Moreover, the approach is less extensible and flexible since the algorithms are hard-coded.

The consideration of high-expressive NFP descriptions and the definition of parametric NFP evaluations are provided only by [24]. The proposed exploitation of axioms support complex and conditioned NFP definitions (e.g., if the client is older than 60 or younger than 10 years old the invocation price is lower than 10 euro). The hybrid approach proposed in this dissertation differs for four different aspects. First, assertions about properties with undefined values are supported by specifying them with a range of possible guaranteed values. This supports the evaluation of offers with some unspecified values. Second, the evaluation of policy/request matching and applicability conditions are decoupled. This supports information retrieval without forcing the requester to know and specify all the information required to evaluate the applicability conditions. In case of incomplete requests, the user is involved to evaluate the actual applicability conditions. Third, as shown in Chapter 4 the PCM-based descriptions used by the hybrid approach can be obtained by wrapping existing specifications defined in several languages. Fourth, the hybrid approach has been tested against a significant set of NFP descriptions.

Chapter 6

Web Service Contract Composition

In the current service composition landscape there is the need to compose different services to provide converged services. The software as a service (SaaS) model envisages a demand-led software market in which businesses assemble and provide services when needed to address a particular requirement. The SaaS vision is a vital contribution to current thinking about software development and delivery that has arisen in part from initiatives in the Web services and electronic-business-communication communities [66].

The SaaS model allows service providers to combine different services, potentially characterized by different service contracts specified by different languages. With the techniques developed so far, it is not so difficult for consumers to compose different services based on published service interfaces. For example, existing platforms like The Process Factory[1] and Boomi[2] provide different connectors for consumers to compose their services from various SaaS providers. However, the consumers need to ensure that the service compositions do not include conflicting service contracts. This assurance cannot be given by a single SaaS provider and currently is not available in existing composition tools.

The usage of each service involved in the composition is defined by a service contract specified with one of the several existing languages (e.g., WSLA, ODRL-S, WSOL, WS-Policy) for service contract description. Even if the service contract heterogeneity can be solved applying the techniques defined in

[1]`http://www.theprocessfactory.com`
[2]`http://www.boomi.com`

Chapter 4, two more problems must be tackled.

The first problem consists in detecting if contracts of services involved in the composition are compatible or not. Incompatibilities among contractual properties (e.g., *QoS*, *Business*, *Service Context* and *License* terms) can determine the impossibility to execute the composite service. As an example, services offering incompatible values in *Data Ownership* (e.g., *copyrighted data* and *free data distribution*) or in *Service Coverage* (e.g., *Europe only* and *US only*) can cause inefficient composite service execution since their data are protected by different rules and their coverage is limited to different world regions.

The second problem consists in creating a unified contract for the composite service that must be defined composing the contracts offered by the services involved in the composition. This unified contract specifies recommended property values, evaluated on the basis of the property values included in the offered contracts. As an example, supposing that compatible values for *Data Ownership* (e.g., *copyrighted data* and *personal data usage*) and *Service Coverage* (e.g., *Europe only* and *Italy only*) are specified, a proper value for these properties must be evaluated and included into the composite service contract.

Past research has neglected contracts of composite services when performing service composition by considering only functional parameters (service interfaces). Furthermore, past research has not focused on tools and algorithms dealing with contract compatibility evaluation when combining different services from different providers. Typically, they deal with only contract negotiation between consumer and service in a point-to-point manner.

Service contract compatibility is strongly dependent on the structure of the composition. This is related to not only control flows but also data flows and composition patterns. While certain works address QoS-based compatibility for control flows, currently there is not a good understanding of how to check contract compatibility for data, the input/output of services, whose contract terms are not always the same to that of the services. Contract terms associated with the use of service and the use of data are different and proper techniques must be defined to evaluate the compatibility for both data and service. Hence, service contract compatibility should be understood in this context.

This dissertation proposes the SeCO$_2$ framework to support service composers to deal with service contracts in service composition. The focus is on techniques used by SeCO$_2$ to evaluate service contract compatibility and to define a contract for the composite service.
The SeCO$_2$ framework is based on the following assumptions:

- functional-based Web service composition has just been performed and a description stating its *control flow* (i.e., the sequence in which the services are invoked) and the *data flow* (i.e., the exchange of data between services) is available;

- the problem related to service contract heterogeneity has just been solved using the the mapping technique defined in Chapter 4. Therefore, service contracts are defined as PCM-based WSML descriptions including NFPs defined using the *SeCO Reference Ontology*.

One of the most innovative characteristics of the SeCO$_2$ framework is how service contract compatibility evaluation and composition are performed. Currently, there is no distinction between the description of properties related to service usage (e.g., *Request Limit*) and properties related to the data produced by the service (e.g., *Data Ownership*). This produces ambiguities in service contract specifications. As an example, the property *Price* can refer to the amount of money needed for invoking a service or it can refer to the amount of money needed for receiving an amount of data from a service. Nevertheless, this distinction is not tackled in current specifications, however, it is very important for the service contract compatibility evaluation and composition. As an example, consider the scenario in which a user wants to create a *Travel Agency Service* (TA) by composing a *Flight Booking Service* (FB), an *Hotel Reservation Service* (HR) and a *Payment Service* (PS). The services follow a sequential execution and data are exchanged between FB and PS and between HR and PS. To evaluate the compatibility of the *availability time range* term (i.e., the time range in which the service is available) in the FB contract, the same term in HR contract must be considered since FB and HR are executed one after the other. Viceversa, to evaluate the compatibility of the *data ownership* term (i.e., a license term stating how the data produced by the service are protected) in the FB contract, the same term in PS contract must be considered since FB data are managed by PS.

The SeCO$_2$ framework performs service contract compatibility evaluation and composition considering both the *control flow* and the *data flow* of the composite service. Furthermore, another characteristics of the service contract compatibility evaluation and composition performed by SeCO$_2$ is that it is not limited to *QoS* but it is also extended to other types of property that can be included in a service contract like *Business*, *Service Context* and *License* terms. Table 6.1 shows the influences between each identified service contract property type and *control* and *data flows*. The scenario is the following: (i) *QoS* are influenced only by *control flow*; (ii) *Service context terms* are independent and (iii) *Business* and *License* terms can be influenced by both the flows.

	control flow	**data flow**	**independent**
Quality of Service (QoS)	X		
Service Context			X
Business	X	X	
License	X	X	

Table 6.1: Data and control flows in contract compatibility evaluation

This chapter proceeds as follows. Section 6.1 provides a detailed description of actors, data and activities involved in the SeCO$_2$ framework. Sections 6.2 and 6.3 present rules and algorithms to be used for service contract compatibility evaluation and composition. Section 6.4 describes the architecture of the SeCO$_2$ Evaluator tool. Finally, experimental tests and comparison with the state of the art are in Sections 6.5 and 6.6.

6.1 The SeCO$_2$ Framework

The SeCO$_2$ framework supports the activities of the following actors:

- *Service Composers*: submit requests for service contract compatibility evaluation and composition specifying composition descriptions in terms of data and control flows.

- *Domain Experts*: provide expertise in service contract compatibility evaluation and composition.

The framework takes as input *composition descriptions* stating services involved in a composition and their data and control flows and produces the following outputs:

- *compatibility results*: representing the results of the service contract compatibility evaluation. They consist in a list of identified incompatibilities among service contract properties.

- *contract recommendations*: representing recommendations for the definition of the contract for the composite service. They consist in a list of suggested values for the properties that can be included in the composite service contract.

In order to produce the mentioned output, SeCO$_2$ utilizes the following resources:

- *SeCO Reference Ontology*: as described in Chapter 4, this extensible ontology contains semantic description of service contract properties, their allowed values, and the relations among them.

- *Contract term knowledge-base*: a repository that contains additional information about properties described in the reference ontology. Among others, this repository contains the specification of the the influences between each service contract property and *control* and *data flows*.

- *SeCO policies*: representing PCM-based WSML descriptions of service contracts including NFPs defined using the *SeCO Reference Ontology*. The *SeCO policies* are stored in the *SeCO Policy Repository*.

- *Compatibility evaluation rules*: rules for checking the compatibility between *SeCO Policies* of services involved in the composition. Each rule specifies how to evaluate the compatibility in term of a specific property defined in the reference ontology. The rules are stored in the *Compatibility Rule Repository*.

- *Composition rules*: rules for the service contract property composition. Each rule specifies how to evaluate a recommended value for a specific property to be included in a composite service contract. The rules are stored in the *Composition Rule Repository*.

The activities that compose the service contract compatibility evaluation and composition process can be divided in: (i) *setup-time* and (ii) *run-time* activities. Set-up time activities (summarized in Figure 6.1) are:

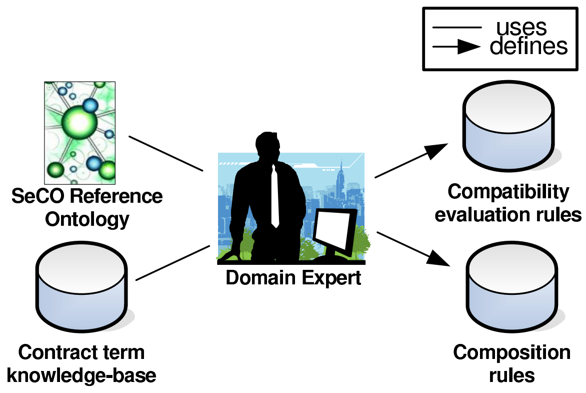


Figure 6.1: SeCO$_2$ setup-time activities

- *Compatibility evaluation rule definition*: for each property specified in the *SeCO reference ontology* a compatibility evaluation rule must be defined. This activity is performed by *Domain Experts* considering data and control flows, as well as composition patterns, that can be associated with a service composition. To perform this activity *Domain Experts* make use of the information available in the reference ontology and in the *Contract term knowledge-base*. Details about this activity are provided in Section 6.2.

- *Composition rule definition*: for each property specified in the *SeCO Reference Ontology* a composition rule must be defined. Also this activity is performed by *Domain Experts* considering data and control flows and using the information available in the reference ontology and in the *Contract term knowledge-base*. Details about this activity are provided in Section 6.3.

Run-up time activities (summarized in Figure 6.2) are the following:

- *Composition description analysis*: aims at analyzing the composition descriptions provided by the *Service Composer* in order to identify the services involved in the composition and data and control flows of the service composition.

- *SeCO policy retrieval*: after having identified the services involved in the composition, their associated *SeCO Policies* are retrieved from the *SeCO Policy Repository*.

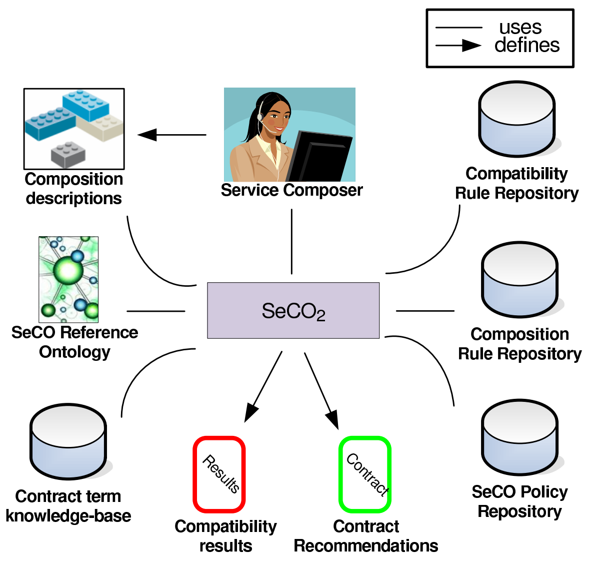


Figure 6.2: SeCO$_2$ run-time activities

- *Service contract compatibility evaluation*: service properties are extracted from each *SeCO Policy*. For each property the related *Compatibility evaluation rule* is extracted from the *Compatibility Rule Repository* and executed considering data and control flows of the service composition. Incompatibilities between service contracts are pointed out to the *Service Composer*. Details about this activity are in Section 6.2.

- *Definition of the contract for the composite service*: for each property included in at least one of the retrieved *SeCO Policies*, the related *Composition rule* is extracted from the *Composition Rule Repository* and executed considering data and control flows of the service composition. The result of each rule is included in the contract for the composite service. Details are in Section 6.3.

6.2 Web Service Contract Compatibility Evaluation

The Web service contract compatibility evaluation supported by the SeCO$_2$ framework accepts a full or part of a full description of service compositions, e.g., the complete structure of a composite service or a workflow region. The used technique in principle can work with any composition language (e.g., BPEL4WS[3]) as long as *control* and *data flows* among services in a composi-

[3]http://www.ibm.com/developerworks/library/specification/ws-bpel/

tion are captured. As stated above, a *data flow* represents dependencies in data provisioning and defines strong dependencies between services (e.g., data produced by serviceA are needed by serviceB to perform its activities), instead a *control flow* represents the execution order of the services in terms of *composition patterns* allowed by the composition language. As an example, the BPEL4WS construction *sequence, switch, while, pick* and *flow* can be easily mapped to three different *composition patterns*: *sequential execution*, *parallel execution* and *conditional execution*. The structure activities *sequence* and *while* are considered as *sequential executions* (i.e., loop can be reduced to sequence composition with unfolding techniques); *switch* and *pick* are *conditional executions*; *flow* is a *parallel execution*.

6.2.1 Contract Compatibility Evaluation Rules

One of the activities to be performed at set-up time is the *Compatibility evaluation rule definition*. As described in Chapter 3, service contract properties can be classified in qualitative and quantitative properties. Moreover, as shown in Table 6.1, properties can be differently influenced by control and data flows. This section starts providing a set of basic compatibility evaluation rules for qualitative and quantitative properties. Then, examples of how *Domain Experts* can used these basic rules to produce compatibility evaluation rules for a specific properties considering data and control flows are shown.

Qualitative properties must be evaluated considering the semantic relations among property values that are defined in the *SeCO Reference Ontology*. Examples of basic *compatibility evaluation rules* for qualitative properties are the following:

- *"Relation-based"* rule: it is applicable to properties assuming values characterized by semantic relations among them. Examples are *partnership* (i.e., values characterized by *partOf* relations) and *subsumption* (i.e., values characterized by *isA* relations). These relations are checked to verify the compatibility among property values.

- *"Compatible value list"* rule: it is applicable to properties assuming a small set of possible values. The compatibility list among these values is stored into the reference ontology by the definition of *isCompatibleWith* relations.

Quantitative properties must be evaluated considering the constraint operators used to specified the offered values. As described in Chapter 3, constraint operators can be *binary* (e.g., $=, \leq, \geq$) or *ternary* (e.g., interval). Examples of basic compatibility evaluation rules for quantitative properties are the following:

- *"Binary operator"* rules: a different rule is defined for each couple of binary constraint operators. As an example, a proper rule must be applied to evaluate the compatibility between two offered values defined using the operators $\leq$ and $\geq$. The results of these rules are numeric values in the range [0..1] stating the degree of compatibility between the two offered values. Details about these rules can be found in Table 5.2 of Chapter 5.

- *"Ternary operator"* rule: it is used to evaluate the overlap between ranges of values associated with comparable properties. Also in this case, the results is a numeric value in the range [0..1] stating the degree of compatibility between the two ranges.

In order to provide examples of specific-property rules that can be defined by *Domain Experts* using the above mentioned basic rules and in order to analyze the impact of data and control flows on the evaluation, a set of properties covering all the NFP types (i.e., QoS, Business, Context and License terms) will be used in the illustrative examples:

- `Service Coverage`: a *Service Context* term defining world regions in which a service is available. This property assumes values characterized by *partnership* relations.

- `Pricing`: a *Business* term describing possible pricing models offered by the service. This property applies to the data delivered by the service.

- `Payment`: a *Business* term describing the cost for the service usage or for data delivered by the service.

- `Scalability`: a *QoS* indicating the maximum number of transactions accepted per minute. It is defined using a binary constrain operator.

- `Request Limit`: a *QoS* indicating the maximum number of request accepted in a day. It is defined using a binary constrain operator.

- `Availability Time Range`: a *QoS* defining the interval of time in which the service can be used (e.g., [8am..6pm]). It is defined using a ternary constrain operator.

- `Data Ownership`: a *License* term stating how the data produced by the service are protected.

- `Permissions`: a *License* term defining how the service can be used and assumes values characterized by subsumption relations.

Property	**Type**	**Data Flow**	**Control Flow**	**Basic Rule**
Service Coverage	Service Context			partnership
Pricing	Business	X		compatible value list
Payment (for data usage)	Business	X		binary
Payment (for service usage)	Business		X	binary
Scalability	QoS		X	binary
Request Limit	QoS		X	binary
Availability Time Range	QoS		X	ternary
Data Ownership	License	X		compatible value list
Permissions	License		X	subsumption

Table 6.2: Examples of associations between properties and basic rules

Table 6.2 shows, for each analyzed properties, its type, the influences with data and control flows, and the basic rule that must be used to define the specific-property compatibility evaluation rule. In the following, an example of logic that can be define for each evaluation rule is provided.

The `Service Coverage` property is independent from data and control flows since its value must be checked in all the contracts of the services involved in the composition. The compatibility among two values is evaluated defining a *"Relation-based"* rule focusing on *partnership* relations. In particular, contracts are compatible if their values are characterized by a partnership relation ($\sqsupseteq$) among them. For example, assume that s_1 delivers in the `Worldwide`, s_2 in `Europe` and s_3 in the `US`. Only the following partnership relations are hold: `Worldwide`$\sqsupseteq$`Europe` and `Worldwide`$\sqsupseteq$`US`. Services s_2 and s_3 cannot be included in the same composition since their provision is limited to different geographical area (i.e., `Europe` and `US` are not characterized by a partnership relation among them).

The compatibility on `Pricing` terms in service contracts is checked considering the data flow. The evaluation is performed defining a *"Compatible value list"* rule stating the compatibility among possible pricing models. For example, `flat rate` is compatible with `pay per use with subscription` but it is incompatible with `free per use`. For example, assume that s_2 needs data from s_1. If s_1 is `free per use` compatibility evaluation is not necessary. Elsewhere, in cases of `flat rate` and `pay per use with subscription` the capability of s_2 to perform a payment and a subscription must be verified.

The `Payment` property can refer to data or service usage. In the former case, the data flow of the service composition must be considered and a *"Binary operator"* rule must be defined. For example, assume that s_2 needs data from s_1. If s_1 specifies a `Payment (for data usage)`, the capability of s_2 to meet the costs must be verified. In the latter case, the control flow must be considered. For example, if s_2 is invoked by s_1 and if s_1 specifies a `Payment (for service usage)`, the capability of s_1 to meet the costs must be verified.

The `Scalability` and `Request Limit` properties are checked considering the *composition patterns* included in the control flow specification and defining a *"Binary operator"* rule. For example, assume that service s_1 and s_2 follow a *sequential execution* and that s_1 and s_2 have `Scalability` $= sc_1$ and `Scalability` $\leq sc_2$ respectively. Services s_1 and s_2 are compatible if $sc_1 \leq sc_2$.

The compatibility with respect to `Availability Time Range` among service contracts must be evaluated considering the control flow of the composition and defining a *"Ternary operator"* rule. The evaluation requires (if available) also to consider the property `Mean Execution Time` stating the mean time between the moment a request is sent to the time that the response has been provided to the requester. Moreover, the evaluation must consider the *composition patterns* included in the control flow specification. For example, assume that service s_1 and s_2 follow a *sequential execution* and that s_1 and s_2 have `Availability Time Range` r_1 and r_2 respectively. Moreover, assume that s_1 is characterized by a `Mean Execution Time` equal to et_1. The compatibility of these services is checked according to the following rule: *services* s_1 *and* s_2 *are compatible if* $(\max(r_1) + et_1) \subseteq r_2$.

The compatibility on `Data Ownership` terms in service contracts is checked considering the data flow. The evaluation is performed defining a *"Compatible value list"* rule stating the compatibility among possible ownership values. For example, `copyrighted` is compatible with `personal use` but it is incompatible with `free distribution`. For example, if s_2 needs data from s_1 and if s_1 delivered `copyrighted` data, the two services are incompatible only if s_2 declares `free distribution` of data.

The property `Permissions` is influenced by the data flow of the service composition. The compatibility among two values is evaluated defining a *"Relation-based"* rule focusing on *subsumption* relation. A couple of services s_1 and s_2 with dependencies among them in the *data flow*, presents compatibility in terms of `Permissions` if $s_1.value \succ s_2.value$.

6.2.2 An Algorithm for Contract Compatibility Evaluation

Let $S = \{s_1, s_2, \cdots, s_m\}$ denote the set of services involved in the composition. Each service is characterized by a service operation associated with one or more SeCO Policies. Let $P(s_i) = \{p_1, p_2, \cdots, p_n\}$ indicate the set of policies associated with service s_i. Each policy is composed of one or more offered properties. Let $PR(p_i) = \{pr_1, pr_2, \cdots, pr_w\}$ be the set of properties offered by policy p_i. Each property is specified by: (i) a `name` stating the related ontological concept; (ii) a `type` defining if the property is *CF-inf* (i.e., influence the *control flow*), *DF-inf* (i.e., influence the *data flow*) or *F-ind* (i.e., flow independent); (iii) an `operator`; (iv) a `value` and (v) a `unit` of measure.

Let $CF(s_i) = \{cf_1, cf_2, \cdots, cf_m\}$ denote the *control flow* where each cf_j in $CF(s_i)$ specifies the *composition pattern* between s_i and s_j. Possible values are *sequential, parallel* and *conditional execution*. Let $DF(s_i) = \{df_1, df_2, \cdots, df_m\}$ denote the *data flow* where each df_j in $DF(s_i)$ specifies if there is a dependency in data provisioning between s_i and s_j.

Our service contract compatibility algorithm is listed in Algorithm 1. The algorithm evaluates the compatibility among all the policies of all the couples of services available in the composition. Line 3 defines $\Omega(s_i, s_j)$ as a set of triples. Each triple will contain a policy p_w associated with s_i, a policy p_z associated with s_j, and the result of the compatibility evaluation $\lambda(p_w, p_z)$ among them. The evaluation of $\lambda(p_w, p_z)$ starts in Line 7 defining $\Upsilon(p_w, p_z)$ as a set of comparable properties $[pr_1, pr_2]$ specified in p_w and p_z. $\Upsilon(p_w, p_z)$ is populated by the `Matching` procedure (Line 8) that applies matching rules similar to the ones shown in Section 5.1.1. For each identified couple $[pr_1, pr_2]$ of comparable properties, the algorithm retrieves the related evaluation rule using the procedure `Extract` and specifying the property *name* (Line 10). As stated above, at this point the evaluation proceeds considering the property `type`. If the property is *CF-inf* then procedure `EvalRuleF` is invoked specifying the retrieved rule, the two comparable properties $[pr_1, pr_2]$ and the *control flow* information about the services s_i, s_j that offer the properties (Line 12). If the property is *DF-inf* then the same procedure `EvalRuleF` is invoked but

Algorithm 1 Compatibility Evaluation

```
1: for all $s_i \in S$ do
2:   for all $s_j \in S(j \neq i)$ do
3:     $\Omega(s_i, s_j) = \phi$ where $\Omega(s_i, s_j)$ is a set of triples $[p_w, p_z, \lambda(p_w, p_z)]$
4:     for all $p_w \in P(s_i)$ do
5:       for all $p_z \in P(s_j)$ do
6:         $\lambda(p_w, p_z) = \phi$, where $\lambda(p_w, p_z)$ is a set of triples $[pr_i, pr_j, result]$
7:         $\Upsilon(p_w, p_z) = \phi$, where $\Upsilon(p_w, p_z)$ is a set of comparable properties $[pr_1, pr_2]$
8:         $\Upsilon(p_w, p_z) = Matching(p_w, p_z)$
9:         for all $[pr_1, pr_2] \in \Upsilon(p_w, p_z)$ do
10:          $rule = Extract(pr_1.name)$
11:          if $pr_1.type =' CF - inf'$ then
12:            $\lambda(p_w, p_z) = \lambda(p_w, p_z) \cup EvalRuleF(rule, pr_1, pr_2, cf_j \in CF(s_i))$
13:          else
14:            if $pr_1.type =' DF - inf'$ then
15:              $\lambda(p_w, p_z) = \lambda(p_w, p_z) \cup EvalRuleF(rule, pr_1, pr_2, df_j \in DF(s_i))$
16:            else
17:              $\lambda(p_w, p_z) = \lambda(p_w, p_z) \cup EvalRule(rule, pr_1, pr_2)$
18:            end if
19:          end if
20:        end for
21:        $\Omega(s_i, s_j) = \Omega(s_i, s_j) \cup [p_w, p_z, \lambda(p_w, p_z)]$
22:      end for
23:    end for
24:  end for
25: end for
```

specifying the *data flow* information about the services s_i, s_j (Line 15). Finally, if the property is *F-ind* then the procedure `EvalRule` that does not consider composition flows is invoked (Line 17). The result of the evaluation is saved in $\lambda(p_w, p_z)$ that contains the evaluation of all the comparable properties in p_w and p_z. Finally, the triple $[p_w, p_z, \lambda(p_w, p_z)]$ is saved in $\Omega(s_i, s_j)$ (Line 21) that contains the evaluation for all the policies offered by s_i and s_j.

6.3 Web Service Contract Recommendation for Service Composition

After the evaluation of compatibility between service contracts involved in a composition, the next step consists in recommending property values to be included in the contract for the composite service. The recommendations are evaluated applying *Composition rules* on comparable property values available in the compatible service contracts.

6.3.1 Contract Composition rules

Each service contract property is associated with a composition rule. This section starts providing examples of basic rules to be used by *Domain Experts* to define composition rules for qualitative and quantitative properties. Then, examples of how to used these basic rules considering data and control flows, as well as composition patterns are shown.
Examples of rules for qualitative properties are the following:

- *"OR"-rule* : if at least one of the services involved in the composition owns the property, also the composite service will own it.
- *"AND"-rule* : if all the services involved in the composition owns the property, also the composite service will own it.
- *"Constraining Value"-rule* : is applied for properties assuming values characterized by relations (e.g., subsumption($\succ$), partnership ($\sqsubseteq$)). The most constraining value among the ones offered by the service will be included in the contract for the composite service.

The presence of an undefined value (i.e., a service contract involved in the composition does not specify the value of the considered property) does not modify the result of the *"OR"-rule*. The *"AND"-rule* and *"Constraining Value"-rule* consider undefined values as if the service does not own the property and their results is the absence of the property also in the contract of the composite service.

Quantitative properties are managed using the following rules. The presence of undefined value determines an undefined value in the contract of the composite service.

- *"MIN"-rule* : the composite service offers the minimum among the values offered by the services involved in the composition.
- *"MAX"-rule* : the composite service offers the maximum among the values offered by the services involved in the composition.
- *"SUM"-rule* : the composite service offers the sum of the values offered by each service involved in the composition.
- *"AVG"-rule* : the composite service offers the mean of the values offered by each service involved in the composition.

- *"MUL"-rule* : the composite service offers a value determined multiplying the values offered by each service involved in the composition.

- *"UNION"-rule* : is used for properties expressed as a range of values. The composite service offers a range of values determined by the union of the ranges offered by each service involved in the composition.

- *"INTERSECTION"-rule* : is used for properties expressed as a range of values. The composite service offers a range of values determined by the intersection of the ranges offered by each service involved in the composition.

In order to provide examples of specific-property rules that can be defined by *Domain Experts* using the above mentioned basic rules and in order to analyze the impact of data and control flows, as well as composition patterns on the evaluation, the same set of properties used in Section 6.2.1 will be used in the following illustrative examples.

`Service Coverage`, `Data Ownership`, `Pricing` and `Permissions` are qualitative properties characterized by different influences with data and control flows of the service composition. The composition rules for these properties are defined using the basic *"Constraining Value"-rule*.

The composition rule for `Service Coverage` consists in identifying the most constraining value in terms of *partnership relations* ($\sqsubseteq$) among all the values included in all the contracts of the service involved in the composition. Examples of *partnership relations* among values assumed by this property are `Worldwide`$\sqsubseteq$`Europe`$\sqsubseteq$`Italy` where `Italy` is the most constraining value.

For what regard `Data Ownership` and `Pricing`, the composition rules consist in identifying the most constraining value in terms of *subsumption relations* ($\succ$) among all the values included in all the contracts of the service specified in the data flow of the composition. An example of *subsumption relation* among `Data Ownership` values is `Copyrighted`$\succ$`Personal-use` where `Copyrighted` is the most constraining value. An example of *subsumption relation* among `Pricing` values is `Pay-per-use-with-subscription`$\succ$`Flat-rate` where `Pay-per-use-with-subscription` is the most constraining value.

The same composition rule is used for `Permissions` but, for this property, the contracts of the service specified in the control flow of the composition are considered. Examples of *subsumption relations* among `Permissions` values is

`Adaptation`$\succ$`Composition`$\succ$`Derivation` where `Adaptation` is the most constraining value.

As state above, the evaluation of recommended property values for the composite service contract can be influenced by the composition patterns included into the data and control flows of the service composition. Several properties are associated with composition rules stating how to use the basic rules for each composition patterns that can be used to represent the data and control flows.

In order to understand the logic of pattern-based composition rules, let consider the problem to compose values of `Payment (for data usage)` (PDU), `Scalability` (SC), `Request Limit` (RL) and `Availability Time Range` (ATR) specified in contracts offered by three services *s1*, *s2* and *s3*. As shown in Table 6.2, PDU is influenced by the data flow instead SC, RL and ATR are influenced by the control flow. Let consider the following compositions where *s3* produces data representing the output of the service composition:

- (C1): *s1*, *s2* and *s3* follow a sequential execution. Data are sent from *s1* to *s2* and from *s2* to *s3*.
- (C2): *s1* and *s2* are executed in parallel. Data are sent from *s1* and *s2* to *s3*.
- (C3): *s1* and *s2* represents alternative services (i.e., conditional execution). Data are sent from *s1* or *s2* to *s3*

The composite service contract (CSC) for C1 is defined as follow:

- PDU(CSC)=SUM(PDU(s1),PDU(s2),PDU(s3))
- SC(CSC)=SC(s1)
- RL(CSC)=RL(s1)
- ATR(CSC)=ATR(s1)

The composite service contract (CSC) for C2 is defined as follow:

- PDU(CSC)=SUM(PDU(s1),PDU(s2),PDU(s3)
- SC(CSC)=[MIN(SC(s1),SC(s2))..MAX(SC(s1),SC(s2))]
- RL(CSC)=[MIN(RL(s1),RL(s2))..MAX(RL(s1),RL(s2))]

- ATR(CSC)=INTERSECTION(ATR(s1)+ATR(s2))

In the case of C3 the composite service will be characterized by two different conditioned service contracts CSC1 and CSC2:

- if (s1 and s3 are used)
  - PDU(CSC1)=SUM(PDU(s1),PDU(s3))
  - SC(CSC1)=SC(s1)
  - RL(CSC1)=RL(s1)
  - ATR(CSC1)=ATR(s1)
- if (s2 and s3 are used)
  - PDU(CSC2)=SUM(PDU(s2),PDU(s3))
  - SC(CSC2)=SC(s2)
  - RL(CSC2)=RL(s2)
  - ATR(CSC2)=ATR(s2)

6.3.2 An Algorithm for Contract Composition

Algorithm 2 Contract Composition

```
1:  for all $c_i \in C$ do
2:      $CSC(c_i) = \phi$
3:      $\Upsilon(c_i) = Matching(c_i)$ where $\Upsilon(c_i)$ is a set of $[pr.name, valueList]$
4:      for all $v_j \in \Upsilon(c_i)$ do
5:          for all $pr_z.name \in v_j$ do
6:              $rule = Extract(pr_z.name)$
7:              if $pr_z.type =' CF - inf'$ then
8:                  $CSC(c_i) = CSC(c_i) \cup EvalCompRuleF(rule, valueList_j, CF(c_i))$
9:              else
10:                 if $pr1.type =' DF - inf'$ then
11:                     $CSC(c_i) = CSC(c_i) \cup EvalCompRuleF(rule, valueList_j, DF(c_i))$
12:                 else
13:                     $CSC(c_i) = CSC(c_i) \cup EvalCompRule(rule, valueList_j)$
14:                 end if
15:             end if
16:         end for
17:     end for
18: end for
```

Let $C = \{c_1, c_2, \cdots, c_m\}$ denotes the set of possible service contract compositions determined combining the compatible contracts offered by each service involved in the composition. Let each property offered by a service contract be specified by: (i) a `name` stating the related ontological concept; (ii) a `type` defining if the property is *CF-inf* (i.e., influence the *control flow*), *DF-inf* (i.e., influence the *data flow*) or *F-ind* (i.e., flow independent); (iii) an `operator`; (iv) a `value` and (v) a `unit` of measure. Let $CSC(c_i)$ denote the composite service contract related to the service contract composition c_i.

The service contract composition algorithm is listed in Algorithm 2. The algorithm proposes a service contract for each possible combination of the compatible contracts offered by the services involved in the composition. Line 3 defines $\Upsilon(c_i)$ as a set of v_j that are couple [*pr.name*, *valueList*] where *valueList* contains all the values for the property identified by *pr.name* offered by the contracts in c_i. $\Upsilon(c_i)$ is populated by the `Matching` procedure that applies matching rules such as the ones shown in Section 5.1.1. For each $pr.name$ included in $\Upsilon(c_i)$, the algorithm retrieves the related composition rule using the procedure `Extract` (Line 6).

At this point the composition proceeds considering the property `type`. If the property is *CF-inf* then procedure `EvalCompRuleF` is invoked specifying the retrieved rule, the *valueList* of the $pr.name$ and the *control flow* of the service composition (Line 8). If the property is *DF-inf* then the same procedure `EvalCompRuleF` is invoked but specifying the *data flow* (Line 11). Finally, if the property is *F-ind* then the procedure `EvalCompRule` that does not consider composition flows is invoked (Line 13). The result of the evaluation is saved in $CSC(c_i)$.

6.4 The Architecture of the SeCO$_2$ Evaluator tool

In order to support *Service Composers* and *Domain Experts* in the setup-time and run-time activities mentioned in Section 6.1, the architecture shown in Figure 6.3 is proposed for the SeCO$_2$ Evaluator tool. The same approach used to define the architecture of the PoliMaR tool (see Section 5.2) is adopted.

The architecture of the SeCO$_2$ Evaluator tool is composed of independent modules that supply services through an API that gives access to:

- a `Knowledge Manager`, which is in charge of storing *SeCO Policies* into the `SeCO Policy Repository` and of managing the data stored in the `Contract term knowledge-base`;

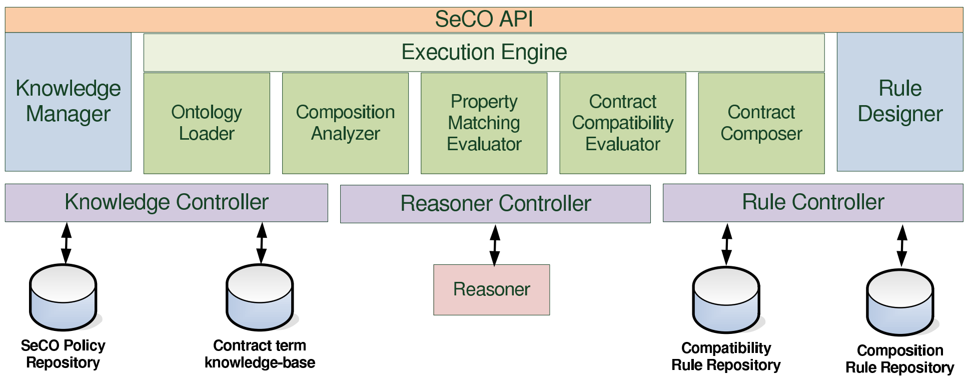


Figure 6.3: The architecture of the SeCO$_2$ Evaluator tool

- an `Execution Engine`, which receives *Composition descriptions* submitted by *Service Requestors* and implements the execution strategies to fulfill the service contract compatibility evaluation and composition process;

- a `Rule Designer`, which allows *Domain Experts* to define *Compatibility evaluation rules* and *Composition rules*.

The SeCO$_2$ components rely on common services (namely, `Knowledge Controller`, `Reasoner Controller` and `Rule Controller`). These services have the primary task of decoupling SeCO$_2$ from any specific platform or technology for data storage and reasoning activities. The core components of the SeCO$_2$ Evaluator tool are the following:

- `Ontology Loader`: is in charge of loading all the knowledge necessary for the contract compatibility evaluation and composition process into the `Reasoner` through the `Reasoner Controller`. The knowledge consists in the *SeCO Policies* stored in the `SeCO Policy Repository`.

- `Composition Analyzer`: is used for analyzing *Composition descriptions* and extracting data such as: (i) the services involved in the composition; (ii) the *data flow* and (iii) the *control flow*.

- `Property Matching Evaluator`: implements the process necessary to identify comparable contract properties. This component is the same used into the PoliMaR tool (see Chapter 5 for details).

- `Contract Compatibility Evaluator`: implements the process related to the compatibility evaluation among contract property values. It retrieves the *Compatibility evaluation rules* related to each property identified by the `Property Matching Evaluator` from the `Compatibility Rule Repository` through the `Rule Controller`. The algorithm used by this component to perform the compatibility evaluation is the one described in Section 6.2.2.

- `Contract Composer`: implements the service contract composition process retrieving the *Composition rules* related to each property identified by the `Property Matching Evaluator` from the `Composition Rule Repository` through the `Rule Controller`. The algorithm used by this component to perform the composition is the one described in Section 6.3.2.

Currently, only the core components of the SeCO Evaluator tool have been implemented using Java JDK 1.6.0 update 3. The *SeCO Policies* are represented in the WSML language. The Wsml2Reasoner API (v0.6.1) are used to communicate with the KAON2 reasoner. This preliminary version of the tool is available at http://sourceforge.net/projects/seco2/.

6.5 Experimental Results

In order to demonstrate the efficiency of the proposed approach for contract compatibility evaluation and composition, the case study described in Section 2.2.4 is considered.

Assume that a service composer provides two different composition structures, namely composition (a) and composition (b) (see Figure 2.1 in Chapter 2), for `Purchase Data Analysis (PDA)` service characterized by different control flow, data flow and composition patterns among the following Web services: (i) *Yahoo! Shopping Web Service*[4] selected as `Merchant Validation Service` (MVS); (ii) *XWebCheckOut Web Service*[5] as `Purchase Processing Service` (PPS); (iii) *Aivea Shipping Web Service*[6] as `Freight Transportation Service` (FTS); (iv) *ValidateCreditCard Web Service*[7] as

[4]http://developer.yahoo.com/shopping/V1/merchantSearch.html
[5]http://www.xwebservices.com/Web_Services/XWebCheckOut/
[6]http://www.aivea.com/shipping-web-service.htm
[7]http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=14

`Payment Verification Service` (PS) and (v) *DOTS Lead Validation Web Service*[8] as `Purchase Validation Service` (PVS). The focus of this experiment is on service contracts. Thus, assume that these Web services match the functionalities required for the `PDA` service.

The selected Web services are characterized by service contracts available only as HTML texts in their Websites (i.e., ODRL-S, WSLA and WSOL specifications are not available). Moreover, these contracts are unclear, ambiguous and limited to few information. This forces the service consumer to manually compare them and, often, further information from the service providers are needed. Modeling and mapping techniques presented in Chapter 4 are not applicable due to the absence of structured specifications. In order to overcome this strong limitation, SeCO Policies are produced using information described in the HTML texts and inserting realistic properties in case of limited descriptions. These SeCO Policies are summarized in Table 6.3.

For each selected Web service, the properties `Service Coverage`, `Data Ownership`, `Request Limit`, `Pricing (for data usage)` and `Scalability` described in Section 6.2.1 are considered.

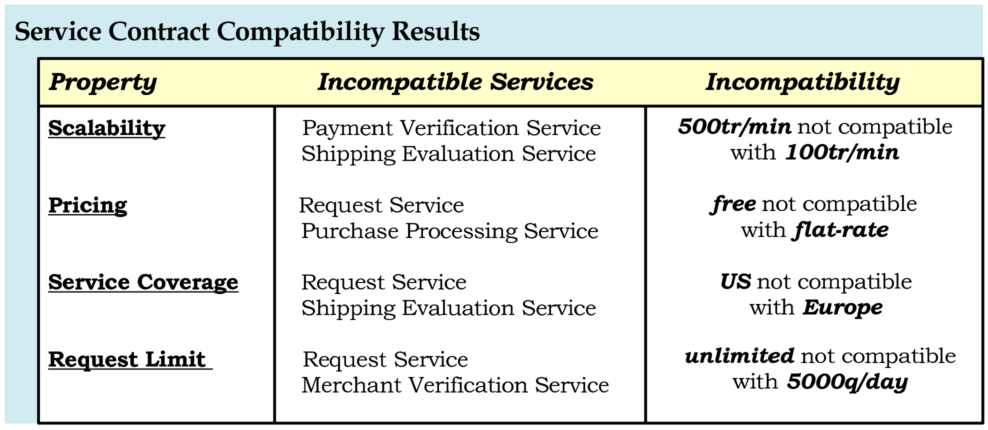
	Serv.Cov.	**Data Own.**	**Request Limit**	**Pricing**	**Scalability**
Request Service (RS)	US	personal-use	unlimited	free	100tr/min
Yahoo! Shopping (MVS)	Worldwide	copyrighted	5000q/day	free	100tr/min
XWebCheckOut (PPS)	Worldwide	free-distrib.	unlimited	flat-rate	100tr/min
Aivea Shipping (FTS)	Europe	free-distrib.	unlimited	flat-rate	100tr/min
ValidateCreditCard (PS)	Worldwide	free-distrib.	unlimited	free	500tr/min
DOTS Lead Valid. (PVS)	Worldwide	free-distrib.	unlimited	free	500tr/min

Table 6.3: SeCO Policies offered by services involved in the composition

Applying evaluation rules described in Section 6.2, the compatibility evaluation results for composition (a) and composition (b) produced by our SeCO$_2$ framework are given in Figures 6.4 and 6.5. The following results must be underlined:

- incompatibility on `Service Coverage` is found in both the compositions since the property is independent from data and control flows;
- incompatibility on `Pricing` is found in both the compositions because both are characterized by a data flow from RS and PPS;
- incompatibility on `Request Limit` is found in both the compositions but between different services. This is determined by the different data flows

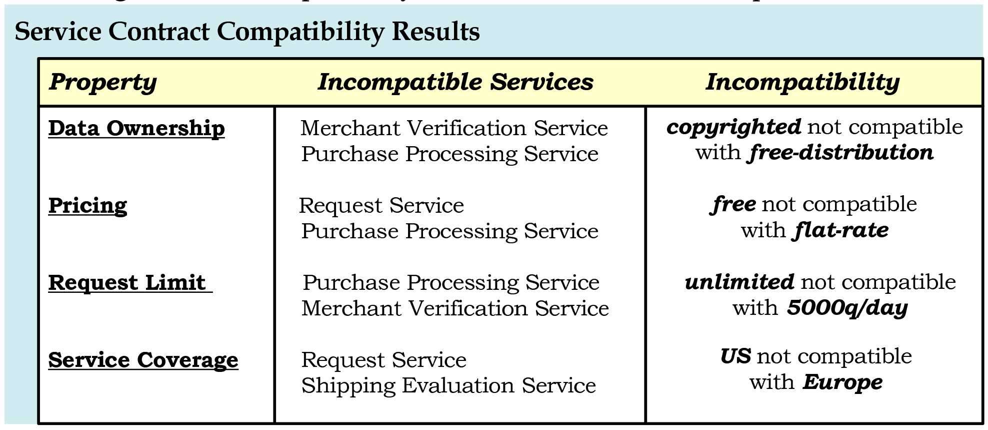
[8]http://www.serviceobjects.com/products/composite/lead-validation



Service Contract Compatibility Results

Property	*Incompatible Services*	*Incompatibility*
Scalability	Payment Verification Service Shipping Evaluation Service	***500tr/min*** not compatible with ***100tr/min***
Pricing	Request Service Purchase Processing Service	***free*** not compatible with ***flat-rate***
Service Coverage	Request Service Shipping Evaluation Service	***US*** not compatible with ***Europe***
Request Limit	Request Service Merchant Verification Service	***unlimited*** not compatible with ***5000q/day***

Figure 6.4: Compatibility evaluation results for composition (a)



Service Contract Compatibility Results

Property	*Incompatible Services*	*Incompatibility*
Data Ownership	Merchant Verification Service Purchase Processing Service	***copyrighted*** not compatible with ***free-distribution***
Pricing	Request Service Purchase Processing Service	***free*** not compatible with ***flat-rate***
Request Limit	Purchase Processing Service Merchant Verification Service	***unlimited*** not compatible with ***5000q/day***
Service Coverage	Request Service Shipping Evaluation Service	***US*** not compatible with ***Europe***

Figure 6.5: Compatibility evaluation results for composition (b)

involving MVS;

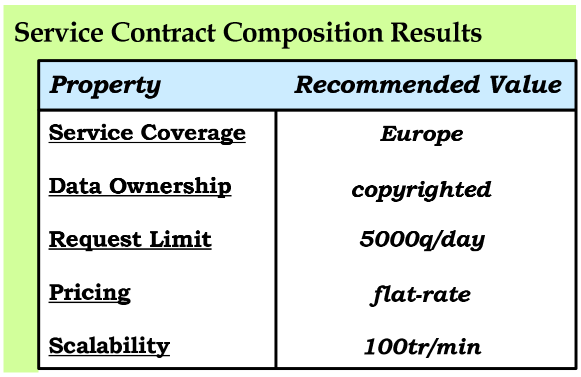
- incompatibility on `Scalability` is found only in `composition a`. This result depends on the different control flows (i.e., in composition (a) FTS is invoked after PS instead in composition (b) it is invoked after PPS);

- incompatibility on `Data Ownership` is found only in composition (b). This result depends on the different data flows (i.e., in composition (a) MVS data are managed by RS instead in composition (b) they are managed by PVS).

Let now focus on service contract composition. Suppose that the identified incompatibilities have been solved and that the new compatible contracts are the ones in Table 6.4. In order to define the composite service contracts, the rules described in Section 6.3.1 are used.

	Ser.Cov.	Data Own.	Request Limit	Pricing	Scalability
Request Service (RS)	Worldwide	personal-use	5000q/day	flat-rate	100tr/min
Yahoo! Shopping (MVS)	Worldwide	copyrighted	5000q/day	free	100tr/min
XWebCheckOut (PPS)	Worldwide	personal-use	unlimited	flat-rate	100tr/min
Aivea Shipping (FTS)	Europe	personal-use	unlimited	flat-rate	100tr/min
ValidateCreditCard (PS)	Worldwide	personal-use	unlimited	free	100tr/min
DOTS Lead Valid. (PVS)	Worldwide	personal-use	unlimited	free	500tr/min

Table 6.4: Compatible contracts offered by the services



Service Contract Composition Results

Property	*Recommended Value*
Service Coverage	*Europe*
Data Ownership	*copyrighted*
Request Limit	*5000q/day*
Pricing	*flat-rate*
Scalability	*100tr/min*

Figure 6.6: Recommended composite service contract

The recommended composite service contract is shown in Figure 6.6. The same result is obtained for composition (a) and composition (b):

- *Europe* is the recommended value for the property `Service Coverage` since it represents the most constraining value in terms of *partnership relation* ($\sqsubseteq$) among the offered values (i.e., *Worldwide*$\sqsubseteq$*Europe*).

- *Copyrighted* is the recommended value for the property `Data Ownership` since it represents the most constraining value in terms of *subsumption relation* ($\succ$) among the values offered by the service included in the data flow of the service composition (*Copyrighted*$\succ$*Personal-use*).

- *5000q/day* is the recommended value for the property `Request Limit` since it is the value of the *Request Service (RS)* that starts the control flow of the service composition.

- *Flat-rate* is the recommended value for the property `Pricing` since it represents the most constraining value in terms of *subsumption relation*

($\succ$) among the values offered by the service included in the data flow of the service composition.

- *100tr/min* is the recommended value for the property `Scalability` since it is the value of the *Request Service (RS)* that starts the control flow of the service composition.

6.6 Related Works

Until now there is no work dealing with service contract compatibility and composition evaluation considering data and control flows of the service composition.

The most cited works [9–11, 67, 68] related to contract-based service composition reduce the problem to the evaluation of QoS constraints among composite services and user requirements. The AgFlow framework [9] evaluates the QoS of composite services with an extensible multidimensional quality model and considering the control flow of the service composition. The evaluation is made in order to optimize the QoS of the composite service starting from user requirements and candidate component services. Other examples of QoS-based composition are in [10] that aims at defining composition rules to evaluate global values of QoS dimensions according to specific workflow patterns. The constraint-driven Web service composition tool presented in [11] reduces the service composition problem to a constraint satisfaction problem focusing on business and process constraints. Planning and pricing algorithms are presented in [67, 68]. The focus is on defining dynamic functions for price minimization that account for the structure of the service composition. An extension to QoS evaluation based on the definition of utility functions is also presented. The limitation of [9–11, 67, 68] is that they consider only a small set of service contract terms (i.e., QoS). The evaluation of qualitative properties (e.g., license terms) is not tackled. Moreover, they assume that property descriptions are always available and specified using a common language.

Other papers [41–43] related to contract-based service composition deal with service customization. Customization capability (i.e., the possibility to customize functional, non-functional and behavioral service aspects) through wrappers and XML-based mechanisms to enable Web service customization are proposed in [41] and [42] respectively. An approach and a tool for a systematic and WS-compliant policy customization for SaaS service consumption is presented in [43]. The proposed innovative idea consists in allowing service providers to describe the customizability of a SaaS service in a standard format which becomes the basis for service consumers to provide inputs to customize the service. These inputs are checked to ensure its compliance within the service customization capability defined by providers. The limitation of these approaches for service customization is that the compliance on service properties is checked between a service provider and a service consumer while the compliance between multiple contracts defined on possibly different set of properties is not tackled.

Past research has neglected the importance of data flow in contract-based service composition. A contribution limited to the importance of data flow dependencies into efficient behavioral analysis is in [69]. This paper presents an approach to raise the precision of established verification techniques for BPEL4WS by analyzing the data flow dependencies to avoid false-positive error warnings and to prove the compliance of a BPEL process w.r.t. given rules and regulations.

The problem regarding the management of heterogeneity service contracts in service composition is tackled in [37] and [18]. The work in [37] underlines that matching SLA templates represent an unrealistic assumption in system where service consumer and provider meet dynamically and on-demand. The proposed VieSLAF framework allows for the management of SLA mappings defined as XSLT documents. Thus, these mappings allow partner with slightly different templates to negotiate with each other and increase the number of potential negotiation partner. The work in [18] proposes an approach to describing business services, the creation of SLA templates from Universal Service Description Language (USDL) [70] service descriptions and the negotiation of SLAs is proposed in [18]. Services are created using Integrated Service Engineering (ISE) workbench that implements a model-driven approach to service development and support the management of SLA templates. The VieSLAF framework and the ISE workbench represent innovative but partial solutions since only the management of service level agreement (SLA) mappings is supported.

Chapter 7

Conclusion and Future Extensions

Web services promise universal interoperability and integration of services developed by independent providers to execute business processes by discovering and composing services distributed over the Internet. This means that the key to build complex and valuable processes among cooperating organizations relies on the efficiency of discovering appropriate Web services and composing them. The increasing availability of Web services that offer similar functionalities requires mechanisms that go beyond the pure functional discovery and composition of Web services.

A possible solution consists in enhancing Web service discovery and composition with the evaluation of contracts that define non-functional properties (NFPs) and applicability conditions associated with a Web service. The Web service discovery activity can be enriched with a Web service *selection* phase in which discovered services are evaluated in order to identify the one(s) that offers the contract including the set of NFPs that better fulfills the NFP requirements specified by the user. The Web service composition activity can be extended with a new phase dealing with the evaluation of compatibility among contracts offered by the discovered services [9–11].

Despite of the enhancement of Web service discovery and composition with the evaluation of Web service contracts is recognized as a promising solution for the automatic definition of valuable business processes [9, 21], there is the lack of tools and algorithms providing a complete solution.

This dissertation has proposed solutions to the following problems:

The Web service contract specification problem

Problem Definition: existing languages lack in the required expressivity to fully describe different types of NFPs concerning a variety of service aspects (e.g., *Quality of Service (QoS)*, *Business*, *Service Context* and *License* terms) that can be specified in a service contract.
Proposed Solution: this dissertation proposes the Policy-Centered Metamodel (PCM) as a solution to *the Web service contract specification problem*. The PCM allows for the description of expressive NFPs to be included in service contracts. The PCM refers to a full set of constraint operators (e.g., $\leq$, $\geq$, interval, exist, include) and *relevance* attributes for sophisticated descriptions of quantitative and qualitative NFP offers and requests. Moreover, the PCM allows clustering of NFPs in sets (i.e., *policies*) in order to capture technological and business interdependencies among properties. Experimental results show the feasibility to define PCM-based Web service contract specifications for the logistic operator domain.

The Web service contract heterogeneity problem

Problem Definition: the lack of standard languages determines heterogeneity in service contract specifications. This means that service consumers and providers can specify their service contracts as they wish, raising interoperability issues due to the impossibility to perform automatic matching when multiple services referring to heterogeneous service contracts are to be processed (e.g., in a composition).
Proposed Solution: this dissertation proposes a solution to *the Web service contract heterogeneity problem* that consists in using the WSML formalization of the Policy-Centered Metamodel (PCM) for modeling rich NFP descriptions to be included in service contracts. Moreover, the proposed techniques to perform the mapping of existing service contracts defined in different languages (WSLA, WS-Policy, ODRL-S, WSOL) to PCM-based WSML descriptions have shown the feasibility to reuse available specifications.

The Web service contract selection problem

Problem Definition: current approaches to improve Web service discovery with a selection phase based on the evaluation of NFPs specified in Web service contracts present some weaknesses. Non-semantic approaches (e.g., [22, 23]) are characterized by low precision due to only syntactic NFP descriptions. Semantic approaches (e.g., [21, 24]) allows for detailed descriptions but present performance problems due to current limitations of semantic tools when dealing with reasoning tasks.

Proposed Solution: this dissertation proposes the Policy Matchmaker and Ranker (PoliMaR) tool as a solution to *the Web service contract selection problem*. The PoliMaR tool allows for an effective and flexible Web service contract selection using an hybrid approach that combines logic-based and algorithmic techniques. The tool overcomes the limitations of purely semantic approaches using logical reasoning techniques on semantic Web service contract descriptions only when they are strictly necessary to improve the precision of the selection. Moreover, the PoliMaR tool outperforms current approaches offering high levels of: (i) *expressivity*, by supporting PCM-based WSML descriptions; (ii) *generality*, by allowing semantic-based mediation between NFP descriptions based on multiple ontologies through the definition of logical axioms and rules; (iii) *extensibility*, by supporting the customization of functions to be used for NFP evaluation and (iv) *flexibility*, by allowing the selection process also with unspecified/missing values.

Experimental results on the PoliMaR tool demonstrate that the performance problems of semantic tools can be limited by the introduction of caching techniques. These techniques allows the PoliMaR to be used as a Web application since its execution time is comparable to the time required by traditional discovery/ranking engines available on the Internet.

The Web service contract composition problem

Problem Definition: incompatibilities among *QoS*, *Business*, *Service Context* and *License* terms specified in contracts of services involved in the composition can determine the impossibility to execute the composite service. Existing composition tools do not support the evaluation of Web service contract compatibility when combining different services from different providers.

Proposed Solution: this dissertation proposes the Service Contract Compatibility ($SeCO_2$) Evaluator tool as a solution to *the Web service contract composition problem*. The $SeCO_2$ Evaluator supports service contract compatibility evaluation considering both the *control flow* (i.e., the sequence in which the services are invoked) and the *data flow* (i.e., the exchange of data between services) of the service composition. Experimental results demonstrates the effectiveness of the proposed solution.

7.1 Future Extensions

Several possible extensions for the solutions proposed in this dissertation can be identified. Among others, the most relevant and challenging are the following:

Dynamic Property Management

The definition of run-time measurement functions and the possibility to include them in Web service contract specifications would extend the current support for dynamic NFP evaluations given by the solutions proposed in this dissertation. The Policy-Centered Metamodel (PCM) supports the specification of *dynamic* NFPs by representing contractual terms that are strictly related to the context of execution and cannot be stated at publishing time. In a PCM-based Web service contract specification, a dynamic NFP is defined through list/range of possible values the NFPs can assume at execution-time. Despite of that, the PCM does not focus on the measurement of the NFP values at run-time. The introduction of NFP measurement functions is challenging since dependencies among NFPs and the invocation of third-party services must be considered. Past research [9] focused on the definition of functions for run-time QoS measurements. Less attention has been given to the run-time measurements of *Business*, *Context* and *License* terms that often required data from third-party services. As an example, a service able to evaluate the distance between two addresses is required to measure the *price* for a freight transportation since it depends on the distance between the pick-up and the delivery locations. Moreover, a currency converter service can be required in order to make offered and requested price comparable.

Advanced Web Service Contract Selection

A more sophisticated approach to evaluate satisfaction degrees between offered and requested NFPs along the Web service contract selection should be defined.

The PCM provides a set of quantitative constraint operators that allows for a great deal of expressiveness in requested NFP specifications (see Chapter 3). This supports the definition of the Web service contract selection as a Constraint Satisfaction Problem (CSP), in which requested NFPs are modelled as vectors of constraints (i.e., constrain hierarchies) where each constraint is associated with an *utility function* [71]. These functions assign utility assessment to every value in the requested interval, so that the higher the assessment, the better the consideration for the users.

On the other hand, PCM supports the specification of offered NFPs as a ranges into which NFP values would probably fall since it is impractical to expect that Web services always perform ideally as they are declared. This means that offered NFPs can be associated with certain *probabilistic distribution functions* (e.g., *normal, uniform, exponential distributions*).

The possibility of providing CSP solutions addressing *utility functions* for required NFPs and *probabilistic distribution functions* for offered NFPs can be further investigated. Preliminary results on the *advanced Web service contract selection* are presented in [72] and [73].

Web Service Contract Monitoring

This dissertation provides tools (i.e., the PoliMaR and the SeCO$_2$ Evaluator tools) for Web service contract selection and composition as setup-time activities supporting the definition of valuable business processes. The need to support Web service contract management at run-time is only partially covered: once the contract has been established, there is no way to monitor the actual process execution to verify the fulfillment of the contractual clauses. A complete solution should include monitoring facilities. As described in [70], this task consists in collecting information necessary to realize the execution of tradeable services with respect to established contracts and to get usage data relevant for billing.

While the collection of monitoring data is a continuous process supported by service and system monitoring tools, a parallel activity to find out the interesting events and correlations is needed in order to determine the fulfilment of the Web service contract. The definition of tools and algorithms for Web service contract violation prevention is a research open problem.

Definition of Lightweight Web Service Contract Descriptions

Ontology-based frameworks, such as WSMO and OWL-S, provide for rich and formalized descriptions of semantic Web services that enhance traditional Web service discovery and composition processes. Experimental results demonstrate that semantic tools available today show unsatisfactory performance dealing with reasoning tasks on these descriptions. Current research focuses on the definition of a lightweight solution for semantic Web service descriptions that consists in defining semantically annotated WSDL documents. As an example, the Semantic Annotations for WSDL (SAWSDL) [59] is a light semantic annotation language that allows for attaching semantic to the principal elements within a WSDL document, simply by annotating them with concepts from a semantic model (e.g., classes within an OWL ontology).

A possible extension of this dissertation consists in defining techniques for annotating WSDL documents with lightweight contractual term descriptions. Since a WSDL document describes different service layers (e.g., *service, operation, I/O message*), techniques to associate proper contractual terms with each layer can be defined. As an example, the WSDL *service* element should be annotated with *Context* terms defining service profile aspects (e.g., *Service Coverage*) instead *I/O message* element should be annotated with *License* terms defining conditions on data usage (e.g., *Data Ownership*).

Appendix A

PCM Ontology

This appendix includes the WSML specification of the Policy-Centered Metamodel (PCM) described in Chapter 3 and the PCM-based WSML specification of the *silver policy* described in Chapter 2.

Listing A.1: the Policy-Centered Metamodel

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml−syntax/wsml−flight"
namespace {
_"http://www.siti.disco.unimib.it/research/ontologies/pcm#",
wsml _"http://www.wsmo.org/wsml/wsml−syntax#"}

ontology pcm


/*Definition of concepts included in the Policy Nfp Hierarchy*/

concept PolicyNfp
    hasExpression ofType (1 1) PolicyExpression

concept QualitativeNfp subConceptOf PolicyNfp
    hasExpression ofType (1 1) QualitativeExpression

concept SetNfp subConceptOf QualitativeNfp
    hasExpression ofType (1 1) SetExpression

concept CustomNfp subConceptOf QualitativeNfp
    hasExpression ofType (1 1) CustomExpression

concept QuantitativeNfp subConceptOf PolicyNfp
    hasExpression ofType (1 1) QuantitativeExpression

concept SingleValueNfp subConceptOf QuantitativeNfp
    hasExpression ofType (1 1) SingleValueExpression
```

```
concept RangeNfp subConceptOf QuantitativeNfp
       hasExpression ofType (1 1) RangeExpression

/*Definition of concepts included in the Nfp Expression
Hierarchy*/

concept NfpExpression
       hasOperator ofType (1 1) ConstraintOperator

concept QualitativeExpression subConceptOf PolicyExpression
       hasOperator ofType (1 1) QualitativeOperator

concept SetExpression subConceptOf QualitativeExpression
       hasOperator ofType (1 1) SetOperator

concept CustomExpression subConceptOf QualitativeExpression
       hasOperator ofType (1 1) CustomOperator

concept QuantitativeExpression subConceptOf PolicyExpression
       hasOperator ofType (1 1) QuantitativeOperator
       hasUnit ofType (1 1) Unit

concept SingleValueExpression subConceptOf QuantitativeExpression
       hasOperator ofType (1 1) BinaryOperator
       hasParameter ofType (1 1) _decimal

concept RangeExpression subConceptOf QuantitativeExpression
       hasOperator ofType (1 1) TernaryOperator
       hasMinParameter ofType (1 1) _decimal
       hasMaxParameter ofType (1 1) _decimal


/*Definition of concepts included in the Constraint Operator
Hierarchy*/

concept ConstraintOperator

concept QuantitativeOperator subConceptOf ConstraintOperator

concept QualitativeOperator subConceptOf ConstraintOperator

concept SetOperator subConceptOf QualitativeOperator

concept CustomOperator subConceptOf QualitativeOperator

concept BinaryOperator subConceptOf QuantitativeOperator

concept TernaryOperator subConceptOf QuantitativeOperator


/*Definition of the concepts Policy and RequestedPolicy*/
```

```
concept Policy
      nonFunctionalProperties
            dc#label hasValue "NFP Policy"
            dc#description hasValue "A policy collecting a number of NFP specifications"
      endNonFunctionalProperties
      hasServiceReference ofType _string
      hasConditionDescription ofType (0 1) _string
      hasCondition impliesType PolicyCondition
      hasNfp impliesType (0 *) PolicyNfp


concept RequestedPolicy subConceptOf Policy
      hasNfp impliesType Request


/*Definition of concepts included in the Request Hierarchy*/

concept Request subConceptOf PolicyNfp
      hasRelevance ofType (1 1) _decimal

concept SetRequest subConceptOf {SetNfp, Request}

concept CustomRequest subConceptOf {CustomNfp, Request}

concept SingleValueRequest subConceptOf {SingleValueNfp, Request}

concept RangeRequest subConceptOf {RangeNfp, Request}


/*Definition of general concepts and relations*/

concept Unit

concept PolicyCondition

concept Client

relation satisfiedFor (ofType PolicyCondition, ofType Client)

relation satisfied (ofType PolicyCondition)


/*Instances of constraint operator concepts*/

instance greaterEqual memberOf BinaryOperator
      nonFunctionalProperties
            dc#description hasValue "the required value is a lower bound and the higher offered value is
                  better"
      endNonFunctionalProperties

instance lessEqual memberOf BinaryOperator
      nonFunctionalProperties
```

```
          dc#description hasValue "the required value is an upper bound and the lower offered value
               is better"
     endNonFunctionalProperties

instance atLeast memberOf BinaryOperator
     nonFunctionalProperties
          dc#description hasValue "the required value is a lower bound and the lower offered value is
               better"
     endNonFunctionalProperties

instance atMost memberOf BinaryOperator
     nonFunctionalProperties
          dc#description hasValue "the required value is a upper bound and the higher offered value
               is better"
     endNonFunctionalProperties

instance interval memberOf TernaryOperator
     nonFunctionalProperties
          dc#description hasValue "the required value is specified in a range"
     endNonFunctionalProperties

instance equal memberOf BinaryOperator

instance all memberOf SetOperator

instance exist memberOf SetOperator

instance include memberOf SetOperator


/*Definition of relations supported by the PCM*/

relation composedOf/2

relation transitiveComposedOf/2

axiom composedOfTransitivity
    definedBy
        transitiveComposedOf(?X,?Y):− composedOf(?X,?Y).

axiom composedOfTransitivity2
    definedBy
        transitiveComposedOf(?X,?Z):− transitiveComposedOf(?X,?Y) and transitiveComposedOf(?Y
             ,?Z).

relation partOf/2

relation transitivePartOf/2

axiom partOfTransitivity
    definedBy
        transitiveComposedOf(?X,?Y):− composedOf(?X,?Y).
```

```
axiom partOfTransitivity2
     definedBy
          transitivePartOf(?X,?Z):- transitivePartOf(?X,?Y) and transitivePartOf(?Y,?Z).

axiom partOfComposedOf
     definedBy
          partOf(?X,?Y):- composedOf(?Y,?X).

axiom transitivePartOfComposedOf
     definedBy
          transitivePartOf(?X,?Y):- transitiveComposedOf(?Y,?X).
```

Listing A.2 presents the WSML PCM-based specifications of the *silver policy*.

Listing A.2: SilverPolicy

```
ontology SilverPolicy

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance silverPolicy memberOf pcm#Policy

     pcm#hasServiceReference hasValue _"http://www.siti.disco.unimib.it/research/WSOmniTransport
            #WSFastTransport"

     pcm#hasNfp hasValue offServicePrice2
     pcm#hasNfp hasValue offPaymentDeadline2
     pcm#hasNfp hasValue offHoursToDelivery2
     pcm#hasNfp hasValue offPaymentMethod2
     pcm#hasNfp hasValue offInsurance2


instance offServicePrice2 memberOf nfpo#ServicePrice
     hasExpression hasValue offServicePriceExp2

instance offServicePriceExp2 memberOf nfpo#ServicePriceExpression
     hasOperator hasValue pcm#equal
     hasParameter hasValue 150
     hasUnit hasValue nfpo#euro


instance offPaymentDeadline2 memberOf nfpo#PaymentDeadline
```

```
    hasExpression hasValue offPaymentDeadlineExp2

instance offPaymentDeadlineExp2 memberOf
nfpo#PaymentDeadlineExpression
    hasOperator hasValue pcm#equal
    hasParameter hasValue 60
    hasUnit hasValue nfpo#days


instance offHoursToDelivery2 memberOf nfpo#HoursToDelivery
    hasExpression hasValue offHoursToDeliveryExp2

instance offHoursToDeliveryExp2 memberOf
nfpo#HoursToDeliveryExpression
    hasOperator hasValue pcm#equal
    hasParameter hasValue 24
    hasUnit hasValue nfpo#hours


instance offPaymentMethod2 memberOf nfpo#PaymentMethod
    hasExpression hasValue offPaymentMethodExp2

instance offPaymentMethodExp2 memberOf
nfpo#PaymentMethodExpression
    hasOperator hasValue pcm#all
    hasParameters hasValue {nfpo#carriagePaid,nfpo#carriageForward}


instance offInsurance2 memberOf nfpo#Insurance
    hasExpression hasValue offInsuranceExp2

instance offInsuranceExp2 memberOf nfpo#InsuranceExpression
    hasOperator hasValue pcm#all
    hasParameters hasValue ins#FireInsurance
```

Appendix B

PoliMaR Test Ontologies

This appendix presents excerpts of the WSML ontologies used for the experimental activities on the PoliMaR tool presented in Section 5.2.2 and Section 5.3.1. Listing B.1 represents the *RequestedPolicy* LORequest1 used for test 2,3,5,7,8.

Listing B.1: LORequest1

```
ontology Request

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance LORequest1 memberOf pcm#RequestedPolicy

          pcm#hasNfp hasValue reqBasePrice
          pcm#hasNfp hasValue reqPaymentDeadline
          pcm#hasNfp hasValue reqHoursToDelivery
          pcm#hasNfp hasValue reqPaymentMethod
          pcm#hasNfp hasValue reqInsurance

instance reqBasePrice memberOf nfpo#BasePriceRequest
     hasExpression hasValue reqBasePriceExp
     hasRelevance hasValue 0.8

instance reqBasePriceExp memberOf nfpo#BasePriceExpression
     hasOperator hasValue pcm#lessEqual
     hasParameter hasValue 120
     hasUnit hasValue nfpo#euro
```

```
instance reqPaymentDeadline memberOf nfpo#PaymentDeadlineRequest
    hasExpression hasValue reqPaymentDeadlineExp
    hasRelevance hasValue 0.4

instance reqPaymentDeadlineExp memberOf
nfpo#PaymentDeadlineExpression
    hasOperator hasValue pcm#atLeast
    hasParameter hasValue 15
    hasUnit hasValue nfpo#days


instance reqHoursToDelivery memberOf nfpo#HoursToDeliveryRequest
    hasExpression hasValue reqHoursToDeliveryExp
    hasRelevance hasValue 0.8

instance reqHoursToDeliveryExp memberOf
nfpo#HoursToDeliveryExpression
    hasOperator hasValue pcm#interval
    hasMinParameter hasValue 24
    hasMaxParameter hasValue 48
    hasUnit hasValue nfpo#hour


instance reqPaymentMethod memberOf nfpo#PaymentMethodRequest
    hasExpression hasValue reqPaymentMethodExp
    hasRelevance hasValue 0.6

instance reqPaymentMethodExp memberOf nfpo#PaymentMethodExpression
    hasOperator hasValue pcm#exist
    hasParameters hasValue {nfpo#carriagePaid,nfpo#carriageForward}


instance reqInsurance memberOf nfpo#InsuranceRequest
    hasExpression hasValue reqInsuranceExp
    hasRelevance hasValue 0.6

instance reqInsuranceExp memberOf nfpo#InsuranceExpression
    hasOperator hasValue pcm#include
    hasParameters hasValue ins#FireInsurance
```

Listings B.2,B.3,B.4,B.5,B.6 and B.7 presents the *RequestedPolicies* used in test 4.

Listing B.2: Request1

```
ontology Request1

importsOntology
   {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance LORequest1 memberOf pcm#RequestedPolicy

     pcm#hasNfp hasValue requestedBasePrice1

instance requestedBasePrice1 memberOf nfpo#BasePriceRequest
    pcm#hasExpression hasValue requestedBasePriceExpression1
    pcm#hasRelevance hasValue 0.8

instance requestedBasePriceExpression1 memberOf
nfpo#BasePriceExpression
    pcm#hasOperator hasValue pcm#lessEqual
    pcm#hasParameter hasValue 120
    pcm#hasUnit hasValue nfpo#euro
```

Listing B.3: Request2

```
ontology Request2

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance LORequest2 memberOf pcm#RequestedPolicy

    pcm#hasNfp hasValue requestedPaymentDeadline2
    pcm#hasNfp hasValue requestedBasePrice2

instance requestedPaymentDeadline2 memberOf
nfpo#PaymentDeadlineRequest
    pcm#hasExpression hasValue requestedPaymentDeadlineExpression2
    pcm#hasRelevance hasValue 0.4

instance requestedPaymentDeadlineExpression2 memberOf
             nfpo#PaymentDeadlineExpression
    pcm#hasOperator hasValue pcm#atLeast
    pcm#hasParameter hasValue 15.0
    pcm#hasUnit hasValue nfpo#days

instance requestedBasePrice2 memberOf nfpo#BasePriceRequest
```

```
    pcm#hasExpression hasValue requestedBasePriceExpression2
    pcm#hasRelevance hasValue 0.8

instance requestedBasePriceExpression2 memberOf
nfpo#BasePriceExpression
    pcm#hasOperator hasValue pcm#lessEqual
    pcm#hasParameter hasValue 120
    pcm#hasUnit hasValue nfpo#euro
```

Listing B.4: Request3

```

ontology Request3

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance LORequest3 memberOf pcm#RequestedPolicy

    pcm#hasNfp hasValue requestedPaymentDeadline3
    pcm#hasNfp hasValue requestedBasePrice3
    pcm#hasNfp hasValue requestedHoursToDelivery3

instance requestedPaymentDeadline3 memberOf
nfpo#PaymentDeadlineRequest
    pcm#hasExpression hasValue requestedPaymentDeadlineExpression3
    pcm#hasRelevance hasValue 0.4

instance requestedPaymentDeadlineExpression3 memberOf
            nfpo#PaymentDeadlineExpression
    pcm#hasOperator hasValue pcm#atLeast
    pcm#hasParameter hasValue 15.0
    pcm#hasUnit hasValue nfpo#days

instance requestedBasePrice4 memberOf nfpo#BasePriceRequest
    pcm#hasExpression hasValue requestedBasePriceExpression4
    pcm#hasRelevance hasValue 0.8

instance requestedBasePriceExpression4 memberOf
nfpo#BasePriceExpression
    pcm#hasOperator hasValue pcm#lessEqual
    pcm#hasParameter hasValue 120
    pcm#hasUnit hasValue nfpo#euro

instance requestedHoursToDelivery4 memberOf
nfpo#HoursToDeliveryRequest
```

```
    pcm#hasExpression hasValue requestedHoursToDeliveryExpression4
    pcm#hasRelevance hasValue 0.8

instance requestedHoursToDeliveryExpression4 memberOf
            nfpo#HoursToDeliveryExpression
    pcm#hasOperator hasValue pcm#interval
    pcm#hasMinParameter hasValue 48
    pcm#hasMaxParameter hasValue 72
    pcm#hasUnit hasValue nfpo#hours
```

Listing B.5: Request4

```
ontology Request4

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance LORequest4 memberOf pcm#RequestedPolicy

    pcm#hasNfp hasValue requestedPaymentMethod4
    pcm#hasNfp hasValue requestedPaymentDeadline4
    pcm#hasNfp hasValue requestedBasePrice4
    pcm#hasNfp hasValue requestedHoursToDelivery4

instance requestedPaymentMethod4 memberOf
nfpo#PaymentMethodRequest
    pcm#hasExpression hasValue requestedPaymentMethodExpression4
    pcm#hasRelevance hasValue 0.8

instance requestedPaymentMethodExpression4 memberOf
            nfpo#PaymentMethodExpression
    pcm#hasOperator hasValue pcm#exist
    pcm#hasParameters hasValue nfpo#carriagePaid

instance requestedPaymentDeadline4 memberOf
nfpo#PaymentDeadlineRequest
    pcm#hasExpression hasValue requestedPaymentDeadlineExpression4
    pcm#hasRelevance hasValue 0.4

instance requestedPaymentDeadlineExpression4 memberOf
            nfpo#PaymentDeadlineExpression
    pcm#hasOperator hasValue pcm#atLeast
    pcm#hasParameter hasValue 15.0
    pcm#hasUnit hasValue nfpo#days
```

```
instance requestedBasePrice4 memberOf nfpo#BasePriceRequest
    pcm#hasExpression hasValue requestedBasePriceExpression4
    pcm#hasRelevance hasValue 0.8

instance requestedBasePriceExpression4 memberOf
nfpo#BasePriceExpression
    pcm#hasOperator hasValue pcm#lessEqual
    pcm#hasParameter hasValue 120
    pcm#hasUnit hasValue nfpo#euro

instance requestedHoursToDelivery4 memberOf
nfpo#HoursToDeliveryRequest
    pcm#hasExpression hasValue requestedHoursToDeliveryExpression4
    pcm#hasRelevance hasValue 0.8

instance requestedHoursToDeliveryExpression4 memberOf
            nfpo#HoursToDeliveryExpression
    pcm#hasOperator hasValue pcm#interval
    pcm#hasMinParameter hasValue 48
    pcm#hasMaxParameter hasValue 72
    pcm#hasUnit hasValue nfpo#hours
```

Listing B.6: Request5

```
ontology Request5

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                InsuranceOntology#InsuranceOntology"
    }

instance LORequest5 memberOf pcm#RequestedPolicy

    pcm#hasNfp hasValue requestedPaymentDeadline5
    pcm#hasNfp hasValue requestedInsurance5
    pcm#hasNfp hasValue requestedBasePrice5
    pcm#hasNfp hasValue requestedHoursToDelivery5

instance requestedPaymentDeadline5 memberOf
nfpo#PaymentDeadlineRequest
    pcm#hasExpression hasValue requestedPaymentDeadlineExpression5
    pcm#hasRelevance hasValue 0.4

instance requestedPaymentDeadlineExpression5 memberOf
            nfpo#PaymentDeadlineExpression
    pcm#hasOperator hasValue pcm#atLeast
    pcm#hasParameter hasValue 15.0
    pcm#hasUnit hasValue nfpo#days

instance requestedInsurance4 memberOf nfpo#InsuranceRequest
    pcm#hasExpression hasValue requestedInsuranceExpression5
    pcm#hasRelevance hasValue 0.8

instance requestedInsuranceExpression5 memberOf
nfpo#InsuranceExpression
    pcm#hasOperator hasValue pcm#include
    pcm#hasParameters hasValue ins#FireInsurance

instance requestedBasePrice5 memberOf nfpo#BasePriceRequest
    pcm#hasExpression hasValue requestedBasePriceExpression4
    pcm#hasRelevance hasValue 0.8

instance requestedBasePriceExpression5 memberOf
nfpo#BasePriceExpression
    pcm#hasOperator hasValue pcm#lessEqual
    pcm#hasParameter hasValue 120
    pcm#hasUnit hasValue nfpo#euro

instance requestedHoursToDelivery5 memberOf
nfpo#HoursToDeliveryRequest
    pcm#hasExpression hasValue requestedHoursToDeliveryExpression5
    pcm#hasRelevance hasValue 0.8

instance requestedHoursToDeliveryExpression5 memberOf
            nfpo#HoursToDeliveryExpression
    pcm#hasOperator hasValue pcm#interval
    pcm#hasMinParameter hasValue 48
    pcm#hasMaxParameter hasValue 72
    pcm#hasUnit hasValue nfpo#hours
```

Listing B.7: Request6

```
ontology Request6

importsOntology
    {
        _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    NFPOntology#NFPOntology",
        _"http://www.siti.disco.unimib.it/research/ontologies/
                    InsuranceOntology#InsuranceOntology"
    }

instance LORequest6 memberOf pcm#RequestedPolicy

    pcm#hasNfp hasValue requestedPaymentMethod6
    pcm#hasNfp hasValue requestedPaymentDeadline6
    pcm#hasNfp hasValue requestedInsurance6
    pcm#hasNfp hasValue requestedBasePrice6
    pcm#hasNfp hasValue requestedHoursToDelivery6

instance requestedPaymentMethod6 memberOf
nfpo#PaymentMethodRequest
    pcm#hasExpression hasValue requestedPaymentMethodExpression6
    pcm#hasRelevance hasValue 0.8

instance requestedPaymentMethodExpression6 memberOf
            nfpo#PaymentMethodExpression
    pcm#hasOperator hasValue pcm#exist
    pcm#hasParameters hasValue nfpo#carriagePaid

instance requestedPaymentDeadline6 memberOf
nfpo#PaymentDeadlineRequest
    pcm#hasExpression hasValue requestedPaymentDeadlineExpression6
    pcm#hasRelevance hasValue 0.4

instance requestedPaymentDeadlineExpression6 memberOf
            nfpo#PaymentDeadlineExpression
    pcm#hasOperator hasValue pcm#atLeast
    pcm#hasParameter hasValue 15.0
    pcm#hasUnit hasValue nfpo#days

instance requestedInsurance6 memberOf nfpo#InsuranceRequest
    pcm#hasExpression hasValue requestedInsuranceExpression6
    pcm#hasRelevance hasValue 0.8

instance requestedInsuranceExpression6 memberOf
nfpo#InsuranceExpression
    pcm#hasOperator hasValue pcm#include
    pcm#hasParameters hasValue ins#FireInsurance

instance requestedBasePrice6 memberOf nfpo#BasePriceRequest
    pcm#hasExpression hasValue requestedBasePriceExpression6
    pcm#hasRelevance hasValue 0.8

instance requestedBasePriceExpression6 memberOf
nfpo#BasePriceExpression
    pcm#hasOperator hasValue pcm#lessEqual
    pcm#hasParameter hasValue 120
    pcm#hasUnit hasValue nfpo#euro

instance requestedHoursToDelivery6 memberOf
nfpo#HoursToDeliveryRequest
    pcm#hasExpression hasValue requestedHoursToDeliveryExpression6
    pcm#hasRelevance hasValue 0.8

instance requestedHoursToDeliveryExpression6 memberOf
            nfpo#HoursToDeliveryExpression
    pcm#hasOperator hasValue pcm#interval
    pcm#hasMinParameter hasValue 48
    pcm#hasMaxParameter hasValue 72
    pcm#hasUnit hasValue nfpo#hours
```

Listings B.8 and B.9 show excerpts of the WSML ontologies representing the *policy repository* and the *NFP ontology* respectively.

Listing B.8: Policy Repository

```
ontology PolicyRepository

importsOntology
  {
    _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
    _"http://www.siti.disco.unimib.it/research/ontologies/
          NFPOntology#NFPOntology",
    _"http://www.siti.disco.unimib.it/research/ontologies/
          InsuranceOntology#InsuranceOntology",
    _"http://www.siti.disco.unimib.it/research/ontologies/
          NFPRepository#NFPRepository"
  }

[...]

instance policy34 memberOf pcm#Policy
  nonFunctionalProperties
    dc#description hasValue "Policiy Instance for Logistic Operator"
  endNonFunctionalProperties

  pcm#hasServiceReference hasValue
      "http://www.siti.disco.unimib.it/research/
      WSEastBulgariaOrdinaryTransport#WSOrdinaryTransport"

  pcm#hasConditionDescription hasValue
      "This Policy is applicable for gold client"

  pcm#hasCondition hasValue policyCondition_policy34

  pcm#hasNfp hasValue nfpr#offeredPaymentMethod0
  pcm#hasNfp hasValue nfpr#offeredBasePrice6
  pcm#hasNfp hasValue nfpr#offeredPaymentDeadline39

  axiom policyConditionAxiom_policy34
    definedBy
      pcm#satisfied(policyCondition_policy34) :-
        client[nfpo#userTypology hasValue gold]
        memberOf nfpo#LogisticOperatorClient.

[...]
```

Listing B.9: NFP Ontology

```
ontology NFPOntology

importsOntology
  {
    _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
    _"http://www.siti.disco.unimib.it/research/ontologies/
          NFPOntology#NFPOntology",
    _"http://www.siti.disco.unimib.it/research/ontologies/
          InsuranceOntology#InsuranceOntology"
  }

instance offeredPaymentMethod0 memberOf nfpo#PaymentMethod
  pcm#hasExpression hasValue offeredPaymentMethodExpression0

instance offeredPaymentMethodExpression0 memberOf
    nfpo#PaymentMethodExpression
  pcm#hasOperator hasValue pcm#all
  pcm#hasParameters hasValue nfpo#carriageForward

instance offeredPaymentMethod1 memberOf nfpo#PaymentMethod
  pcm#hasExpression hasValue offeredPaymentMethodExpression1

instance offeredPaymentMethodExpression1 memberOf
    nfpo#PaymentMethodExpression
  pcm#hasOperator hasValue pcm#all
  pcm#hasParameters hasValue nfpo#carriagePaid

[...]

instance offeredPaymentDeadline6 memberOf nfpo#PaymentDeadline
  pcm#hasExpression hasValue offeredPaymentDeadlineExpression6

instance offeredPaymentDeadlineExpression6 memberOf
    nfpo#PaymentDeadlineExpression
  pcm#hasOperator hasValue pcm#equal
  pcm#hasParameter hasValue 11.0
  pcm#hasUnit hasValue nfpo#days

instance offeredPaymentDeadline7 memberOf nfpo#PaymentDeadline
  pcm#hasExpression hasValue offeredPaymentDeadlineExpression7

instance offeredPaymentDeadlineExpression7 memberOf
    nfpo#PaymentDeadlineExpression
  pcm#hasOperator hasValue pcm#equal
  pcm#hasParameter hasValue 12.0
  pcm#hasUnit hasValue nfpo#days

[...]

instance offeredInsurance5 memberOf nfpo#Insurance
  pcm#hasExpression hasValue offeredInsuranceExpression5
```

```
instance offeredInsuranceExpression5 memberOf
nfpo#InsuranceExpression
  pcm#hasOperator hasValue pcm#all
  pcm#hasParameters hasValue ins#ActOfVandalismInsurance

instance offeredInsurance6 memberOf nfpo#Insurance
  pcm#hasExpression hasValue offeredInsuranceExpression6

instance offeredInsuranceExpression6 memberOf
nfpo#InsuranceExpression
  pcm#hasOperator hasValue pcm#all
  pcm#hasParameters hasValue {ins#DemageInsurance,ins#LossInsurance}

[...]

instance offeredBasePrice15 memberOf nfpo#BasePrice
  pcm#hasExpression hasValue offeredBasePriceExpression15

instance offeredBasePriceExpression15 memberOf
nfpo#BasePriceExpression
  pcm#hasOperator hasValue pcm#equal
  pcm#hasParameter hasValue 200.0
  pcm#hasUnit hasValue nfpo#euro

instance offeredBasePrice16 memberOf nfpo#BasePrice
  pcm#hasExpression hasValue offeredBasePriceExpression16

instance offeredBasePriceExpression16 memberOf
nfpo#BasePriceExpression
  pcm#hasOperator hasValue pcm#equal
  pcm#hasParameter hasValue 210.0
  pcm#hasUnit hasValue nfpo#euro

[...]

instance offeredHoursToDelivery36 memberOf nfpo#HoursToDelivery
  pcm#hasExpression hasValue offeredHoursToDeliveryExpression36

instance offeredHoursToDeliveryExpression36 memberOf
    nfpo#HoursToDeliveryExpression
  pcm#hasOperator hasValue pcm#interval
  pcm#hasMinParameter hasValue 72.0
  pcm#hasMaxParameter hasValue 192.0
  pcm#hasUnit hasValue nfpo#hours

instance offeredHoursToDelivery37 memberOf nfpo#HoursToDelivery
  pcm#hasExpression hasValue offeredHoursToDeliveryExpression37

instance offeredHoursToDeliveryExpression37 memberOf
    nfpo#HoursToDeliveryExpression
  pcm#hasOperator hasValue pcm#interval
  pcm#hasMinParameter hasValue 96.0
  pcm#hasMaxParameter hasValue 228.0
  pcm#hasUnit hasValue nfpo#hours

[...]
```

Listing B.10 represents the WSML specification of the *insurance ontology* used to specified insurance values in the policies included in the *policy repository*. Relations (i.e., *pcm#composedOf*) among insurance values are shown.

Listing B.10: Insurance Ontology

```
ontology InsuranceOntology
  nonFunctionalProperties
    dc#description hasValue "An ontology of insurance"
    dc#subject hasValue "nfp"
    dc#title hasValue "Insurance Ontology"
    dc#date hasValue "2008−09−21"
  endNonFunctionalProperties

importsOntology
  { _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm"}

concept InsuranceType

instance BlanketCover memberOf InsuranceType

instance DamageInsurance memberOf InsuranceType

instance LossInsurance memberOf InsuranceType

instance FireAndTheftInsurance memberOf InsuranceType

instance FireInsurance memberOf InsuranceType

instance TheftInsurance memberOf InsuranceType

instance ActOfVandalismInsurance memberOf InsuranceType

instance ExplosionInsurance memberOf InsuranceType

instance AtmosphericAgentsInsurance memberOf InsuranceType

axiom ax1
  definedBy
    equal(?X,?X):− ?X memberOf ?Z.

axiom ax2
  definedBy
    equal(?X,?Y) :− equal(?Y,?X).
```

```
axiom ax3
  definedBy
    equal(?X,?Z) :− equal(?X,?Y) and equal(?Y,?Z).

axiom ax4
  definedBy
    include(?X,?Y):−
      pcm#transitiveComposedOf(?Y,?X) and
      ?X memberOf InsuranceType and
      ?Y memberOf InsuranceType.

relationInstance pcm#composedOf(BlanketCover, DamageInsurance)

relationInstance pcm#composedOf(BlanketCover, LossInsurance)

relationInstance pcm#composedOf(BlanketCover,
FireAndTheftInsurance)

relationInstance pcm#composedOf(FireAndTheftInsurance,
TheftInsurance)

relationInstance pcm#composedOf(FireAndTheftInsurance,
FireInsurance)

relationInstance pcm#composedOf(FireInsurance,
ActOfVandalismInsurance)

relationInstance pcm#composedOf(FireInsurance, ExplosionInsurance)

relationInstance pcm#composedOf(FireInsurance,
AtmosphericAgentsInsurance)
```

Appendix C

SeCO$_2$ Test Ontologies

This appendix presents excerpts of the WSML ontologies used for the experimental activities on the SeCO$_2$ Evaluator tool presented in Section 6.5. Listing C.1 shows the WSML specification of the *SeCO Reference Ontology* where the concepts related to service contract terms are modelled. Listing C.2 represents the WSML PCM-based specifications of the service contracts offered by the `Request Service` (RS), `Purchase Processing Service` (PPS), `Merchant Validation Service` (MVS), `Payment Verification Service` (PS), `Freight Transportation Service` (FTS) and `Purchase Validation Service` (PVS) selected to compose the `Purchase Data Analysis (PDA)` service.

Listing C.1: SeCO Reference Ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {
_"http://www.siti.disco.unimib.it/research/ontologies/SeCORefOnto#",
      wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
      dc _"http://purl.org/dc/elements/1.1/",
      pcm _"http://www.siti.disco.unimib.it/research/ontologies/pcm#",
      loc _"http://www.cefriel.it/logistic/Location#"}

ontology SeCORefOnto

      nonFunctionalProperties
            dc#creator hasValue "Marco Comerio"
            dc#type hasValue _"http://www.wsmo.org/2004/d2#ontologies"
            dc#subject hasValue "service contract properties"
            wsml#version hasValue "0.1"
            dc#language hasValue "en-UK"
            dc#title hasValue "SeCO Reference Ontology"
            dc#date hasValue "2009-07-01"
```

```
     endNonFunctionalProperties

     importsOntology
          { _"http://www.siti.disco.unimib.it/research/ontologies/pcm#pcm",
          _"http://www.cefriel.it/logistic/Location#Location"}

/*Description of Permissions*/

    concept Permissions subConceptOf pcm#QualitativeNfp
        pcm#hasExpression impliesType (1 1) PermissionExpression

    concept PermissionExpression subConceptOf pcm#QualitativeExpression
        pcm#hasParameters impliesType (1 *) PermissionValue

    concept PermissionValue

    instance adaptation memberOf PermissionValue
    instance composition memberOf PermissionValue
    instance derivation memberOf PermissionValue

/*Description of Availability Time Range*/

    concept AvailabilityTimeRange subConceptOf pcm#QuantitativeNfp
        pcm#hasExpression impliesType (1 1) AvailabilityTimeRangeExpression

    concept AvailabilityTimeRangeExpression subConceptOf pcm#QuantitativeExpression
        pcm#hasUnit impliesType (1 1) TemporalUnit

    concept TemporalUnit subConceptOf pcm#Unit

    instance hour memberOf TemporalUnit
    instance days memberOf TemporalUnit


/*Description of Service Coverage*/

    concept ServiceCoverage subConceptOf pcm#QualitativeNfp
        pcm#hasExpression impliesType (1 1) ServiceCoverageExpression

    concept ServiceCoverageExpression subConceptOf pcm#QualitativeExpression
        pcm#hasParameters impliesType (1 *) loc#Geographical_area

/*Description of Data Ownership*/

    concept DataOwnership subConceptOf pcm#QualitativeNfp
        pcm#hasExpression impliesType (1 1) DataOwnershipExpression

    concept DataOwnershipExpression subConceptOf pcm#QualitativeExpression
        pcm#hasParameters impliesType (1 *) DataOwnershipValue

    concept DataOwnershipValue
    instance personalUse memberOf DataOwnershipValue
    instance copyrighted memberOf DataOwnershipValue
    instance freeDistribution memberOf DataOwnershipValue

/*Description of Scalability*/

    concept Scalability subConceptOf pcm#QuantitativeNfp
        pcm#hasExpression impliesType (1 1) ScalabilityExpression

    concept ScalabilityExpression subConceptOf pcm#QuantitativeExpression
        pcm#hasUnit impliesType (1 1) ScalabilityUnit

    concept ScalabilityUnit subConceptOf pcm#Unit

    instance tr\min memberOf ScalabilityUnit


/*Description of Request Limit. Two subclasses are needed in order
to manage two possible types of parameters a) numeric values and
b) the string "unlimited"*/

    concept RequestLimit subConceptOf pcm#PolicyNfp

    concept RequestLimitNumber subConceptOf RequestLimit
        pcm#hasExpression impliesType (1 1) RequestLimitNumberExpression

    concept RequestLimitNumberExpression subConceptOf pcm#QuantitativeExpression
        pcm#hasUnit impliesType (1 1) RequestLimitUnit

    concept RequestLimitUnit subConceptOf pcm#Unit

    instance query\day memberOf RequestLimitUnit


    concept RequestLimitEnum subConceptOf RequestLimit
        pcm#hasExpression impliesType (1 1) RequestLimitEnumExpression

    concept RequestLimitEnumExpression subConceptOf pcm#QualitativeExpression
        pcm#hasParameters impliesType (1 *) RequestLimitEnumValue

    concept RequestLimitEnumValue

    instance unlimited memberOf RequestLimitEnumValue


/*Description of Pricing. Two subclasses are needed in order to
manage two possible types of parameters /a) numeric values and b)
strings (e.g., free, freeWithSubscription)*/

    concept Pricing subConceptOf pcm#PolicyNfp

    concept PricingNumber subConceptOf {Pricing, pcm#QuantitativeNfp}
```

```
        pcm#hasExpression impliesType (1 1) PricingNumberExpression

    concept PricingNumberExpression subConceptOf pcm#QuantitativeExpression
        pcm#hasUnit impliesType (1 1) PricingUnit

    concept PricingUnit subConceptOf pcm#Unit

    instance euro\year memberOf PricingUnit
    instance USD\year memberOf PricingUnit
    instance euro\month memberOf PricingUnit
    instance USD\month memberOf PricingUnit

    concept PricingEnum subConceptOf {Pricing, pcm#QualitativeNfp}
        pcm#hasExpression impliesType (1 1) PricingEnumExpression

    concept PricingEnumExpression subConceptOf pcm#QualitativeExpression
        pcm#hasParameters impliesType (1 *) PricingEnumValue

    concept PricingEnumValue

    instance freePerUse memberOf PricingEnumValue
    instance payPerUse memberOf PricingEnumValue
    instance freeWithSubscription memberOf PricingEnumValue
    instance flatRate memberOf PricingEnumValue
```

Listing C.2: Policy Repository

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {
_"http://www.siti.disco.unimib.it/research/ontologies/PolicyRepository#",
    pcm _"http://www.siti.disco.unimib.it/research/ontologies/pcm#",
    sro _"http://www.siti.disco.unimib.it/research/ontologies/SeCORefOnto#",
    dc _"http://purl.org/dc/elements/1.1#",
    cou _"http://glue2.cefriel.it/ontologies/countries#"
    }

ontology PolicyRepository

/*Policy offered by the RS*/

    instance policyRS memberOf pcm#Policy

        pcm#hasServiceReference hasValue _"http://www.siti.disco.unimib.it/examples"

        pcm#hasNfp hasValue serCovRS
        pcm#hasNfp hasValue dataOwnRS
        pcm#hasNfp hasValue reqLimRS
        pcm#hasNfp hasValue pricingRS
        pcm#hasNfp hasValue scalabRS
        pcm#hasNfp hasValue timeAvalRanRS
    instance serCovRS memberOf sro#DeliveryLocation
        pcm#hasExpression hasValue serCovRSExpression

    instance serCovRSExpression memberOf sro#DeliveryLocationExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue cou#US

    instance dataOwnRS memberOf sro#DataOwnership
        pcm#hasExpression hasValue dataOwnRSExpression

    instance dataOwnRSExpression memberOf sro#DataOwnershipExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#personalUse

    instance reqLimRS memberOf sro#RequestLimitEnum
        pcm#hasExpression hasValue reqLimRSExpression

    instance reqLimRSExpression memberOf sro#RequestLimitEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#unlimited

    instance pricingRS memberOf sro#PricingEnum
        pcm#hasExpression hasValue pricingRSExpression

    instance pricingRSExpression memberOf sro#PricingEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freePerUse

    instance scalabRS memberOf sro#Scalability
        pcm#hasExpression hasValue scalabRSExpression

    instance scalabRSExpression memberOf sro#ScalabilityExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 100
        pcm#hasUnit hasValue sro#tr\min

/*Policy offered by the MVS*/

    instance policyMVS memberOf pcm#Policy

        pcm#hasServiceReference hasValue _"http://developer.yahoo.com/shopping/V1/
            merchantSearch.html"

        pcm#hasNfp hasValue serCovMVS
        pcm#hasNfp hasValue dataOwnMVS
        pcm#hasNfp hasValue reqLimMVS
        pcm#hasNfp hasValue pricingMVS
        pcm#hasNfp hasValue scalabMVS


    instance serCovMVS memberOf sro#DeliveryLocation
        pcm#hasExpression hasValue serCovMVSExpression
    instance serCovMVSExpression memberOf sro#DeliveryLocationExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue cou#Worldwide

    instance dataOwnMVS memberOf sro#DataOwnership
        pcm#hasExpression hasValue dataOwnMVSExpression

    instance dataOwnMVSExpression memberOf sro#DataOwnershipExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#copyrighted

    instance reqLimMVS memberOf sro#RequestLimitNumber
        pcm#hasExpression hasValue reqLimMVSExpression

    instance reqLimMVSExpression memberOf sro#RequestLimitNumberExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 5000
        pcm#hasUnit hasValue sro#query\day

    instance pricingMVS memberOf sro#PricingEnum
        pcm#hasExpression hasValue pricingMVSExpression

    instance pricingMVSExpression memberOf sro#PricingEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freePerUse

    instance scalabMVS memberOf sro#Scalability
        pcm#hasExpression hasValue scalabMVSExpression

    instance scalabMVSExpression memberOf sro#ScalabilityExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 100
        pcm#hasUnit hasValue sro#tr\min


/*Policy offered by the PPS*/

    instance policyPPS memberOf pcm#Policy

        pcm#hasServiceReference hasValue _"http://www.xwebservices.com/Web_Services/
            XWebCheckOut/"

        pcm#hasNfp hasValue serCovPPS
        pcm#hasNfp hasValue dataOwnPPS
        pcm#hasNfp hasValue reqLimPPS
        pcm#hasNfp hasValue pricingPPS
        pcm#hasNfp hasValue scalabPPS


    instance serCovPPS memberOf sro#DeliveryLocation
        pcm#hasExpression hasValue serCovPPSExpression
    instance serCovPPSExpression memberOf sro#DeliveryLocationExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue cou#Worldwide

    instance dataOwnPPS memberOf sro#DataOwnership
        pcm#hasExpression hasValue dataOwnPPSExpression

    instance dataOwnPPSExpression memberOf sro#DataOwnershipExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freeDistribution

    instance reqLimPPS memberOf sro#RequestLimitEnum
        pcm#hasExpression hasValue reqLimPPSExpression

    instance reqLimPPSExpression memberOf sro#RequestLimitEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#unlimited

    instance pricingPPS memberOf sro#PricingNumber
        pcm#hasExpression hasValue pricingPPSExpression

    instance pricingPPSExpression memberOf sro#PricingNumberExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 100
        pcm#hasUnit hasValue sro#USD\year

    instance scalabPPS memberOf sro#Scalability
        pcm#hasExpression hasValue scalabPPSExpression

    instance scalabPPSExpression memberOf sro#ScalabilityExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 100
        pcm#hasUnit hasValue sro#tr\min


/*Policy offered by the FTS*/

    instance policyFTS memberOf pcm#Policy

        pcm#hasServiceReference hasValue _"http://www.aivea.com/shipping-web-service.htm"

        pcm#hasNfp hasValue serCovFTS
        pcm#hasNfp hasValue dataOwnFTS
        pcm#hasNfp hasValue reqLimFTS
        pcm#hasNfp hasValue pricingFTS
        pcm#hasNfp hasValue scalabFTS


    instance serCovFTS memberOf sro#DeliveryLocation
        pcm#hasExpression hasValue serCovFTSExpression
    instance serCovFTSExpression memberOf sro#DeliveryLocationExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue cou#Europe

    instance dataOwnFTS memberOf sro#DataOwnership
        pcm#hasExpression hasValue dataOwnFTSExpression

    instance dataOwnFTSExpression memberOf sro#DataOwnershipExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freeDistribution

    instance reqLimFTS memberOf sro#RequestLimitEnum
        pcm#hasExpression hasValue reqLimFTSExpression

    instance reqLimFTSExpression memberOf sro#RequestLimitEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#unlimited

    instance pricingFTS memberOf sro#PricingNumber
        pcm#hasExpression hasValue pricingFTSExpression

    instance pricingFTSExpression memberOf sro#PricingNumberExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 49
        pcm#hasUnit hasValue sro#USD\month

    instance scalabFTS memberOf sro#Scalability
        pcm#hasExpression hasValue scalabFTSExpression

    instance scalabFTSExpression memberOf sro#ScalabilityExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 100
        pcm#hasUnit hasValue sro#tr\min


/*Policy offered by the PS*/

    instance policyPS memberOf pcm#Policy

        pcm#hasServiceReference hasValue _"http://www.webservicex.net/WCF/ServiceDetails.aspx?
            SID=14"

        pcm#hasNfp hasValue serCovPS
        pcm#hasNfp hasValue dataOwnPS
        pcm#hasNfp hasValue reqLimPS
        pcm#hasNfp hasValue pricingPS
        pcm#hasNfp hasValue scalabPS


    instance serCovPS memberOf sro#DeliveryLocation
        pcm#hasExpression hasValue serCovPSExpression
    instance serCovPSExpression memberOf sro#DeliveryLocationExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue cou#Worldwide

    instance dataOwnPS memberOf sro#DataOwnership
        pcm#hasExpression hasValue dataOwnPSExpression

    instance dataOwnPSExpression memberOf sro#DataOwnershipExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freeDistribution

    instance reqLimPS memberOf sro#RequestLimitEnum
        pcm#hasExpression hasValue reqLimPSExpression

    instance reqLimPSExpression memberOf sro#RequestLimitEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#unlimited

    instance pricingPS memberOf sro#PricingEnum
        pcm#hasExpression hasValue pricingPSExpression

    instance pricingPSExpression memberOf sro#PricingEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freePerUse

    instance scalabPS memberOf sro#Scalability
        pcm#hasExpression hasValue scalabPSExpression

    instance scalabPSExpression memberOf sro#ScalabilityExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 500
        pcm#hasUnit hasValue sro#tr\min


/*Policy offered by the PVS*/

    instance policyPVS memberOf pcm#Policy

        pcm#hasServiceReference hasValue _"http://www.serviceobjects.com/products/composite/
            lead-validation"

        pcm#hasNfp hasValue serCovPVS
        pcm#hasNfp hasValue dataOwnPVS
        pcm#hasNfp hasValue reqLimPVS
        pcm#hasNfp hasValue pricingPVS
        pcm#hasNfp hasValue scalabPVS

    instance serCovPVS memberOf sro#DeliveryLocation
        pcm#hasExpression hasValue serCovPVSExpression

    instance serCovPVSExpression memberOf sro#DeliveryLocationExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue cou#Worldwide

    instance dataOwnPVS memberOf sro#DataOwnership
        pcm#hasExpression hasValue dataOwnPVSExpression

    instance dataOwnPVSExpression memberOf sro#DataOwnershipExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freeDistribution

    instance reqLimPVS memberOf sro#RequestLimitEnum
        pcm#hasExpression hasValue reqLimPVSExpression

    instance reqLimPVSExpression memberOf sro#RequestLimitEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#unlimited

    instance pricingPVS memberOf sro#PricingEnum
        pcm#hasExpression hasValue pricingPVSExpression

    instance pricingPVSExpression memberOf sro#PricingEnumExpression
        pcm#hasOperator hasValue pcm#all
        pcm#hasParameters hasValue sro#freePerUse

    instance scalabPVS memberOf sro#Scalability
        pcm#hasExpression hasValue scalabPVSExpression

    instance scalabPVSExpression memberOf sro#ScalabilityExpression
        pcm#hasOperator hasValue pcm#equal
        pcm#hasParameter hasValue 800
        pcm#hasUnit hasValue sro#tr\min
```

Bibliography

[1] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B.J. Krämer. Service-oriented computing: A research roadmap. In *Service Oriented Computing (SOC)*, Dagstuhl Seminar Proceedings, 2006.

[2] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-services: Alook behind the curtain. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, San Diego, CA, USA, June 9-12 2003.

[3] Mike P. Papazoglou. Web services and business transactions. *World Wide Web*, 6(1):49–91, 2003.

[4] I. Toma, D. Roman, and D. Fensel. On describing and ranking services based on non-functional properties. In *Proc. of the Third International Conference on Next Generation Web Services Practices (NWESP '07)*, pages 61–66, Washington, DC, USA, 2007. IEEE Computer Society.

[5] H. Kuno G. Alonso, F. Casati and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer, 1 edition, October 2003.

[6] J. Cardoso and A. P. Sheth. Introduction to semantic web services and web process composition. In *Proc. of First Intl Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04)*, San Diego, CA, USA, 2004.

[7] J. O'Sullivan, D. Edmond, and A. Ter Hofstede. What's in a service? towards accurate description of non-functional service properties. *Distributed Parallel Databases*, 12(2-3), 2002.

[8] Mick Kerrigan. Web service selection mechanisms in the web service execution environment (wsmx). In *Proc. of the 21st Annual ACM Symposium on Applied Computing (SAC2006)*, 2006.

[9] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.

[10] M.C. Jaeger, G. Rojec-Goldmann, and G. Muhl. Qos aggregation for web service composition using workflow patterns. In *Proc. of the Enterprise Distributed Object Computing Conference (EDOC '04)*, pages 149–159, Washington, DC, USA, 2004. IEEE Computer Society.

[11] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *Proc. of the 2004 IEEE International Conference on Services Computing (SCC 2004)*, pages 23–30, 2004.

[12] S. Bajaj et Al. *Web Service Policy 1.2 - Framework*. Available at: http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/, April 2006.

[13] G. R. Gangadharan, V. D'Andrea, R. Iannella, and M. Weiss. Odrl service licensing profile (odrl-s). In *Proc. of the 5th International Workshop for Technical, Economic, and Legal Aspects of Business Models for Virtual Goods*, 2007.

[14] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1), 2003.

[15] C. Bussler, D. Fensel, and A. Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Rec.*, 31(4):24–29, 2002.

[16] A. Tsalgatidou and T. Pilioura. An overview of standards and related technology in web services. *Distrib. Parallel Databases*, 12(2-3):135–162, 2002.

[17] H-L Truong, G.R. Gangadharan, M. Treiber, S. Dustdar, and V. D'Andrea. On reconciliation of contractual concerns of web services. In *Proc. of the 2nd Non Functional Properties and Service Level Agreements in SOC Workshop (NFPSLASOC'08)*, Dublin, Ireland, 2008.

[18] J. Spillner, M. Winkler, S. Reichert, J. Cardoso, and A. Schill. Distributed contracting and monitoring in the internet of services. In *Proc. of the*

9th International Conference on Distributed Applications and Interoperable Systems (DAIS), pages 129–142, Lisbon, Portugal, June 9-11 2009.

[19] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. Evolving services from a contractual perspective. In *Proc. of 21st International Conference on the Advanced Information Systems Engineering (CAISE)*, pages 290–304, Amsterdam, The Netherlands, June 8-12 2009.

[20] J. Peer and M. Vukovic. A proposal for a semantic web service description format. In *In proc. of European Conference on Web Services (ECOWS)*, pages 285–299, Erfurt, Germany, September 27-30 2004.

[21] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A qos-aware selection model for semantic web services. In *Proc. of the 4th Intl Conference on Service-Oriented Computing (ICSOC'06)*, pages 390–401, Chicago, IL, USA, 2006.

[22] H-Q Yu and S. Reiff-Marganiec. A method for automated web service selection. In *proc. of the Congress on Services (SERVICES)*, pages 513–520, 2008.

[23] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm. Preference-based selection of highly configurable web services. In *Proc. of the 16th international conference on World Wide Web (WWW '07)*, pages 1013–1022, New York, NY, USA, 2007. ACM.

[24] J. M. Garcia, I. Toma, D. Ruiz, and A. Ruiz-Cortes. A service ranker based on logic rules evaluation and constraint programming. In *Proc. of 2nd Non Functional Properties and Service Level Agreements in SOC Workshop (NFPSLASOC)*, Dublin, Ireland, 2008.

[25] J. O'Sullivan, D. Edmond, and A.H.M. ter Hofstede. Formal description of non-functional service properties. Technical report, Queensland University of Technology, Brisbane, 2005. Available from http://www.service-description.com/.

[26] D. Martin et Al. *Semantic Markup for Web Services*. Available at: http://www.w3.org/Submission/OWL-S/, 2004.

[27] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. D16. 1v0. 2 The Web Service Mod-

eling Language WSML. *WSML Final Draft March Available at: http://www.wsmo.org/TR/d16/d16.1/v0.21/20051005/*, 2005.

[28] K. Kritikos and D. Plexousakis. Semantic qos metric matching. In *Proc. of the European Conference on Web Services (ECOWS)*, pages 265–274, Washington, DC, USA, 2006. IEEE Computer Society.

[29] E. Giallonardo and E. Zimeo. More semantics in qos matching. In *Proc. of International Conference on Service-Oriented Computing and Application (SOCA07)*, pages 163–171, Newport Beach, CA, USA, 2007.

[30] E.M. Maximilien and M.P.Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 08(5):84–93, 2004.

[31] R. Afandi, J. Zhang, and C. A. Gunter. Ampol-q: Adaptive middleware policy to support qos. In *In ICSOC '03: Proceedings of the International Conference on Service Oriented Computing*, pages 165–178, Chicago, IL, USA, 2006.

[32] D.T. Tsesmetzis, I. Roussaki, I.V. Papaioannou, and M.E. Anagnostou. Qos awareness support in web-service semantics. In *Proc. of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW '06)*, pages 128–135, Guadeloupe, French Caribbean, 2006.

[33] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management applications of the web service offerings language (wsol). *Information Systems*, 30(7):564–586, 2005.

[34] N.F. Noy and M.A. Musen. The prompt suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59:2003, 2003.

[35] S. Castano, A. Ferrara, and S. Montanelli. H-match: an algorithm for dynamically matching ontologies in peer-based systems. In *Proc. of the 1st VLDB Int. Workshop on Semantic Web and Databases (SWDB 2003)*, Berlin, Germany, September 2003.

[36] P. Mitra, G. Wiederhold, and S. Decker. A scalable framework for the interoperation of information sources. In *Stanford University*, pages 317–329, 2001.

[37] I. Brandic, D. Music, P. Leitner, and S. Dustdar. Vieslaf framework: Enabling adaptive and versatile sla-management. In *In proc. of International Workshop on Grid Economics and Business Models 2009 (GECON 09)*, pages 60–73, Delft, The Netherlands, 25-28 August 2009.

[38] J. M. Garcia, D. Ruiz, A. Ruiz-Cortes, O. Martin-Diaz, and M. Resinas. An hybrid, qos-aware discovery of semantic web services using constraint programming. In *proc. of the Int. Conf. on Service-Oriented Computing (ICSOC)*, 2007.

[39] S. Chaari, Y. Badr, and F. Biennier. Enhancing web service selection by qos-based ontology and ws-policy. In *proc. of the Symp. on Applied computing (SAC)*, 2008.

[40] Y. Liu, A. Ngu, and L.Z. Zeng. Qos computation and policing in dynamic web service selection. In *Proc. of the 13th international World Wide Web conference on Alternate track papers and posters (WWW Alt. '04)*, pages 66–73, New York, NY, USA, 2004. ACM Press.

[41] J. Rykowski and W. Cellary. Virtual web services: application of software agents to personalization of web services. In *Proc. of 6th International Conference on Electronic Commerce (ICEC 2004)*, pages 409–418, Delft, The Netherlands, October 25-27 2004.

[42] S. Abiteboul, B. Amann, J. Baumgarten, O. Benjelloun, F. D. Ngoc, and T. Milo. Schema-driven customization of web services. In *VLDB*, pages 1093–1096, 2003.

[43] K. Zhang, X. Zhang, W. Sun, H. Liang, Y. Huang, L. Zeng, and X. Liu. A policy-driven approach for software-as-services customization. In *Proc. of 9th IEEE International Conference on E-Commerce Technology (CEC 2007) / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007)*, pages 123–130, Tokyo, Japan, 23-26 July 2007.

[44] F. De Paoli, M. Palmonari, M. Comerio, and A. Maurino. A Meta-Model for Non-Functional Property Descriptions of Web Services. In *Proc. of the IEEE International Conference on Web Services (ICWS)*, Beijing, China, 2008.

[45] M. Comerio, F. De Paoli, M. Palmonari, and I. Toma. Flexible service offering with semantic policies. In *Proc. of Non Functional Properties and Service Level Agreements in Service Oriented Computing (NFPSLASOC)*, Dublin, Ireland, November 12-14 2008.

[46] M. Comerio, H-L Truong, F. De Paoli, and S. Dustdar. Evaluating contract compatibility for service composition in the seco2 framework. In *Proc. of International Conference on Service Oriented Computing (ICSOC)*, Stockholm, Sweden, November 23-27 2009.

[47] M. Comerio. A novel approach to web services discovery. In *Proc. of the 2nd European Young Research Workshop on Service Oriented Computing (YR-SOC)*, Leicester, UK, 2007.

[48] M. Comerio, F. De Paoli, A. Maurino, and M. Palmonari. Nfp-aware semantic web services selection. In *Proc. of the Enterprise Computing Conference (EDOC)*, Annapolis, Maryland, USA, 2007.

[49] G.R. Gangadharan, M. Comerio, H-L Truong, V. D'Andrea, F. De Paoli, and S. Dustdar. License-aware service selection. In *Proc. of the IEEE Conf. on Enterprise Computing, E-Commerce and E-Services (EEE'08)*, 2008.

[50] G.R. Gangadharan, M. Comerio, H-L Truong, V. D'Andrea, F. De Paoli, and S. Dustdar. Lass - license aware service selection: Methodology and framework. In *Proc. of International Conference on Service Oriented Computing (ICSOC)*, Sidney, Australia,, December 1-5 2008.

[51] A. Carenini, D. Cerizza, M. Comerio, E. Della Valle, F. De Paoli, A. Maurino, M. Palmonari, and A. Turati. Glue2: a web service discovery engine with non-functional properties. In *Proc. of the Fifth European Conference on Web Services (ECOWS '07)*, Dublin, Ireland, 2008.

[52] M. Comerio, F. De Paoli, and M. Palmonari. Effective and flexible nfp-based ranking of web services. In *Proc. of International Conference on Service Oriented Computing (ICSOC)*, Stockholm, Sweden, November 23-27 2009.

[53] A. Carenini, D. Cerizza, M. Comerio, E. Della Valle, F. De Paoli, A. Maurino, M. Palmonari, M. Sassi, and A. Turati. Semantic web service discovery and selection: a test bed scenario. In *Proc of. Sixth Intl. Workshop*

on Evaluation of Ontology-based tools and the Semantic Web Service Challenge (EON & SWS-Challenge 2008), 2008.

[54] Hans Wehberg. Pacta sunt servanda. *The American Journal of International Law*, 53(4), 1959.

[55] E. Lupu and M. Sloman. A policy based role object model. In *Proc. of the InternationalEnterprise Distributed Object Computing Conference (EDOC)*, 1997.

[56] Nathan Muller. Managing service level agreements. *International Journal of Network Management*, 9, 1999.

[57] L. Lewis and P. Ray. Service level management definition, architecture, and research challenges. In *Proc. of the Global Telecommunications Conference (GLOBECOM)*, 1999.

[58] G.R. Gangadharan, H-L Truong, M. Treiber, V. D'Andrea, S. Dustdar, R. Iannella, and M. Weiss. Consumer-specified service license selection and composition. In *Proc. of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, pages 194–203, Washington, DC, USA, 2008. IEEE Computer Society.

[59] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[60] N. Rosa, P. R.F. Cunha, L. Freire, and G.R.R. Justo. Process NFL: A language for describing non-functional properties. In *Proc. of the 35th Annual Hawaii International Conference (HICSS)*, pages 3676–3685, Hawaii, USA, March 29 2005.

[61] J. Aagedal, F. Earl, and J. Ecklund. Modelling QoS: Towards a UML Profile. In *Proc. of the 5th International Conference on The Unified Modeling Language (UML '02)*, pages 275–289, London, UK, 2002. Springer-Verlag.

[62] S. Frølund and J. Koistinen. Qml: A Language for Quality of Service Speicifcation. Techincal report, Hewlett Packard, 02 1998. Available from http://www.hpl.hp.com/techreports/98/HPL-98-10.html.

[63] A. Ruiz-Cortes, O. Martin-Diaz, A. D. Toro, and M. Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst.*, 14(4):439–468, 2005.

[64] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. Ws-securitypolicy v1.2,. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.html, 2007.

[65] H-Q Yu and S. Reiff-Marganiec. Non-functional property based service selection: A survey and classification of approaches. In *Proc. of 2nd Non Functional Properties and Service Level Agreements in SOC Workshop (NF-PSLASOC'08)*, Dublin, Ireland, 2008.

[66] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, 2003.

[67] B. Blau, W. Michalk, D. Neumann, and C. Weinhardt. Provisioning of service mashup topologies. In *Proc. of the 16th European Conference on Information Systems (ECIS)*, Galway, Ireland, June 2008.

[68] B. Blau, D. Neumann, C. Weinhardt, and S. Lamparter. Planning and pricing of service mashups. In *Proc. of the 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services (CEC/EEE '08)*, pages 19–26, Washington, DC, USA, 2008. IEEE Computer Society.

[69] S. Moser, A. Martens, K. Görlach, W. Amme, and A. Godlinski. Advanced verification of distributed WS-BPEL business processes incorporating CSSA-based data flow analysis. In *IEEE International Conference on Services Computing (SCC 2007)*, pages 98–105, 2007.

[70] J. Cardoso, M. Winkler, and K. Voigt. A service description language for the internet of services. In *Proc. of the International Symposium on Service Science (ISSS 2009)*, March 2009.

[71] K. Kritikos and D. Plexousakis. Semantic qos-based web service discovery algorithms. In *Proc. of the European Conference on Web Services (ECOWS)*, pages 181–190, 2007.

[72] P. Li, M. Comerio, A. Maurino, and F. De Paoli. An approach of non-functional property evaluation of web services. In *Proc. of International*

Conference on Web Services (ICWS), Los Angeles, CA, USA, July 6-10 2009.

[73] P. Li, M. Comerio, A. Maurino, and F. De Paoli. Advanced non-functional property evaluation of web services. In *Proc. of European Conference on Web Services (ECOWS)*, Eindhoven, The Netherlands, November 9-11 2009.