Model-Driven Semantic Web Service Composition

Roy Grønmo
SINTEF Information and Communication Technology
P.O.Box 124 Blindern, N-0314 Oslo, Norway
Roy.Gronmo@sintef.no

Michael C. Jaeger
TU Berlin, SEK FR6-10, Franklinstrasse 28/29,
D-10587 Berlin, Germany
mcj@cs.tu-berlin.de

Abstract

As the number of available Web services increases there is a growing demand to realise complex business processes by combining and reusing available Web services. The reuse and combination of services results in a composition of Web services that may also involve services provided in the Internet. With semantically described Web services, an automated matchmaking of capabilities can help identify suitable services. To address the need for semantically defined Web services, OWL-S and WSML have been proposed as competing semantic Web service languages. We show how the proposed semantic Web service languages can be utilized within a model-driven methodology for building composite Web services. In addition we combine the semantic-based discovery with the support for processing QoS requirements to apply a ranking or a selection of the candidates. The methodology describes a process which guides the developer through four phases, starting with the initial modelling, and ending with a new composite service that can be deployed and published to be consumed by other users.

1 Introduction

A growing number of Web services are implemented and made available internally in an enterprise or externally for other users to consume. These Web services can be reused and composed in order to realise larger and more complex business processes. We define *Web services* to be services made available by using Internet protocols such as HTTP and XML-based data formats for their description and invocation. Web service registries allows for Web services to be published and discovered. A *Web service composition* is a description of how Web services can interoperate in order to perform a more complex task.

The Web service proposals for description (WSDL), invocation (SOAP) and composition (BPEL4WS) that are most commonly used, lack proper semantic descriptions. This makes it hard to search for appropriate services because a large number of syntactically described services need to be manually investigated to see if they can perform the desired task. In many cases it will also be necessary with additional manual communication between the service requester and the service provider to determine if the provided service can be used by the requester.

Semantically described Web services make it possible to improve the precision of the search for existing services and to automate parts of the service composition process. Two recent proposals have gained a lot of attention: the american-based OWL-S [4] and the european-based Web Services Modelling Language (WSML[1]) [26]. These emerging specifications overlap in some parts and are complementary in other parts.

The leading organization for object-oriented programming, the Object Management Group (OMG), has defined the Unified Modelling Language (UML), a standard graphical language for expressing software models. OMG also promotes a Model-Driven Architecture (MDA) approach for analysis, design and implementation of software systems. In a model-driven development process, models are used to describe business concerns, user requirements, information structures, components and component interactions. These models govern the system development because they can be transformed into executable code.

The work described in this paper adopts the MDA strategy to develop compositions of Web services. We propose a methodology that can be used when modelling semantic Web service compositions. An important question to be addressed in this paper is: *How can the new semantic Web service proposals be utilized within a model-driven Web service composition methodology?* We have identified the following requirements for such a methodology. It shall:

- enable the developer to work with higher-level graphical models instead of lower-level and more verbose lexical counterparts (**Models as the primary artefact**).
- aim to automate large parts of the process by identifying model-driven transformations and automatic steps

[1]Note that WSML is often referred to as WSMO as it is defined by the WSMO working group and the WSML has only recently been published.

that can be implemented by tools (**Automation**).

- utilize the possibilities given by the semantic Web service languages to support a more powerful matchmaking of requirements and capabilities (**Semantic support**).
- utilize Quality-of-Service (QoS) described Web services to process additional criteria for ranking or selection of services. (**QoS support**).

The paper is structured as follows: Section 2 introduces the methodology with its basic principles and how it relates to the Reference Model for Open Distributed Processing (RM-ODP) and the Web services architecture by the W3C; Sections 3 and 4 elaborate the four phases of the methodology; Section 5 describes the automation and iterative aspects of the methodology. Section 6 discusses if our methodology satisfies the above-mentioned requirements; Section 7 covers related work; and finally Section 8 concludes the paper.

2 A Methodology for Discovering and Selecting Services

A fundamental principle of specifying a composite service is to decomposition the main task into a set of identified tasks of less complexity. The composition developer must discover appropriate Web services that may perform the identified tasks. Our methodology considers how semantic and QoS descriptions of both the requested and provided services may be used for identifying the suitable candidates for each of the identified tasks. Before we describe the process of discovery and selection in our methodology, we provide a brief discussion of its basic phases. We consider the Web Service Architecture of the W3C [25] as well as the trading specification as a part of the RM-ODP by the ISO [10] for establishing a defined set of activities that form a trading process:

1. A service provider creates a description about his advertised service. Then he submits the service to a registry or discovery service.
2. A service requester queries the discovery service for a service by submitting a requirement description. The discovery service compares the requirements with the available services and returns the locations of the matched services. For this phase we focus only on determining services that comply functional requirements.
3. In the next phase, the requester selects services from a set of candidates upon selection criteria. In our methodology, we regard the QoS of each candidate as selection criteria. However, other non-functional requirements, such as organisational limitations between service provider or requester could also serve as criteria. The QoS requirements can be used to rank or exclude services. For example, a constraint could define that services are chosen based on their execution time. The final result is an ordered subset of the discovered services from phase two. In the successful case, the subset contains at least one appropriate service for each identified task.
4. The requester processes the set of matching service candidates that also comply with his selection criteria. In our case, the requester can now setup his service composition. Then, the service requester turns into a service provider who advertises and provides the composed service.

In our methodology, we have adopted this basic principle of Web service discovery and selection. This approach enhances the already available discovery methods in the Web services domain.

2.1 Service Registries

To facilitate the discovery, the main proposal in the Web services domain provides a centralised repository specification called the Universal Description Discovery and Integration (UDDI) [24]. Although the UDDI specification has reached its third revision, service discovery is still not commonly used by the Web service developers. Currently, there are no satisfactory products or development methodologies that fully utilize discovery of services over the Internet for Web service composition. Thus we see a motivation for clarifying and improving the discovery of Web services.

Some promising proposals [16] [3] target the fully automatic discovery, configuration and execution of service compositions. Although this is theoretically possible with computer-based reasoning over the semantics of service elements, we do not expect this to be the mainstream reality in the first few years. Our methodology shall therefore still be useful for services that have only limited semantic descriptions or that provide only a syntactic description about its interface. On the other hand, we propose a service composition methodology which is futuristic in the sense that its full potential cannot be met at the time of writing. The largest benefit will first be achieved when there are a very large number of services available which are sufficiently described with semantics and QoS. Furthermore, we propose a registry infrastructure to better support our methodology.

We distinguish between three conceptual types of registries: an *ontology registry*, a *QoS registry* and a general *service registry*. The ontology registry contains ontology concepts and the QoS registry provides the information about the available QoS characteristics. The main registry, the service registry, contains all the published services with at least syntactic information and metadata such as the UDDI proposal. In addition, the services may be semantically annotated and also provide QoS information. Two important invariants are maintained: the ontology concepts associated with published services are synchronized so that they are always available in the Ontology registry, and the QoS characteristics associated with published services are synchronized so that the QoS registry always contains up-to-date information.

The idea of providing semantic and QoS information in a Web service registry is already introduced by other authors. Some proposals [15] extend the UDDI specification with the support for semantic descriptions. For a UDDI repository, proposals exist[18] that process the QoS as an additional criteria for service discovery.

3 Phases 1 and 2: Modelling and Discovery

The service developer performs the modelling of the new composite service as shown in Figure 1. The service developer will search the ontology registry to find appropriate ontology concepts to use as semantic types for the tasks. The result of the search is expected to be a lexical document which can be OWL, WSML etc. We recommend to build transformation tools that automatically imports the lexical representation into a graphical model. The service developer will search the QoS registry to identify QoS concepts that can be used to specify the QoS requirements. Although, there is no de-facto standard yet for representing QoS characteristics, we recommend to build transformation tools that automatically imports the QoS characteristics into a graphical model. The representation of the QoS characteristics in the UML can follow OMG's UML profile for QoS [14]. For the lexical representation of QoS, we propose to consider existing proposals (e.g WSOL [22]). If a specific application case requires further coverage, we must either extend the current proposed languages or design a new language.

Figure 1 shows how UML 2.0 can be used to design the abstract composition model. UML activity models are used to capture the control and data flow of the composition, and class models are used to model the ontology concepts and the QoS characteristics. The abstract composition model is abstract in the sense that no concrete Web services are chosen. The composition is modelled with tasks, represented by UML activities, for which suitable Web services that can perform the tasks are identified later. In this simple example, there are two tasks to be executed in a sequence. The first task is to search for a book by providing an author and a title. The identified book is then used to buy a book in the second task. The second task also requires credit card information as input and returns a receipt. UML 2.0 activity models support modelling of the boundary of the composite task itself. This surrounding boundary has its own input and output parameters which comprises the new operation signature together with the name of the composite task.

One candidate for capturing the domain ontologies in UML is by following the UML profile defined by Djuric [5]. The ontology concepts are grouped by packages, where a package represents a domain ontology. A tagged value not shown in the figure provides a URI link to the domain ontology. Each ontology concept is an OntClass stereotyped class. This UML profile can represent an OWL ontology fully in UML, meaning that the ontology concepts can be precisely defined with attributes and relationships. The ontology concepts are used for specifying the parameter types by performing an annotation to the data objects with semantically defined types. In cases where no suitable domain concepts are found for a data object, the modeller has two alternatives. The first alternative is to associate a data object with a syntactic type such as the XML Schema built-in types (string, integer etc.). In the example this is done for the title parameter of the composite service and the Search for book task. The other alternative is to define a new ontology concept which can be entirely new, or a subtype of an already existing domain concept. If a concept is defined which does not relate to an existing classification system or taxonomy, the developer cannot expect to find published services that match this concept. Still, a new subtype allows for some reasoning that can be utilized in a matchmaking process [12].

Each task in the example of Figure 1 is assigned a category by a note with the stereotype Category. Such a note has four predefined variables and identifies the classification system and the chosen category within this system. In our example, both tasks share the same category. QoS requirements may be captured in a note stereotyped as QoS-Requirements. In our example the QoS requirements note is not attached to any specific task, meaning that the requirements apply to the aggregated values for the whole Web service composition. The QoS requirement in our examples states that the total monthly subscription price of all the used services should be at most 10 Euros. We have described how to express QoS requirements in UML models in more detail in our preceding paper [6]. The semantic annotation and the QoS requirement of the abstract model are used in the subsequent phases to search for candidate services and finally select the concrete services that realize each task in the abstract composition. The outcome of the first phase is an abstract composition model that contains all the needed information for performing service discovery and selection.

The second phase handles discovery of suitable Web services. The discovery process is based on matchmaking of semantic descriptions. We assume that a Web service registry is available with the following information provided for each Web service: a) a service interface description, and

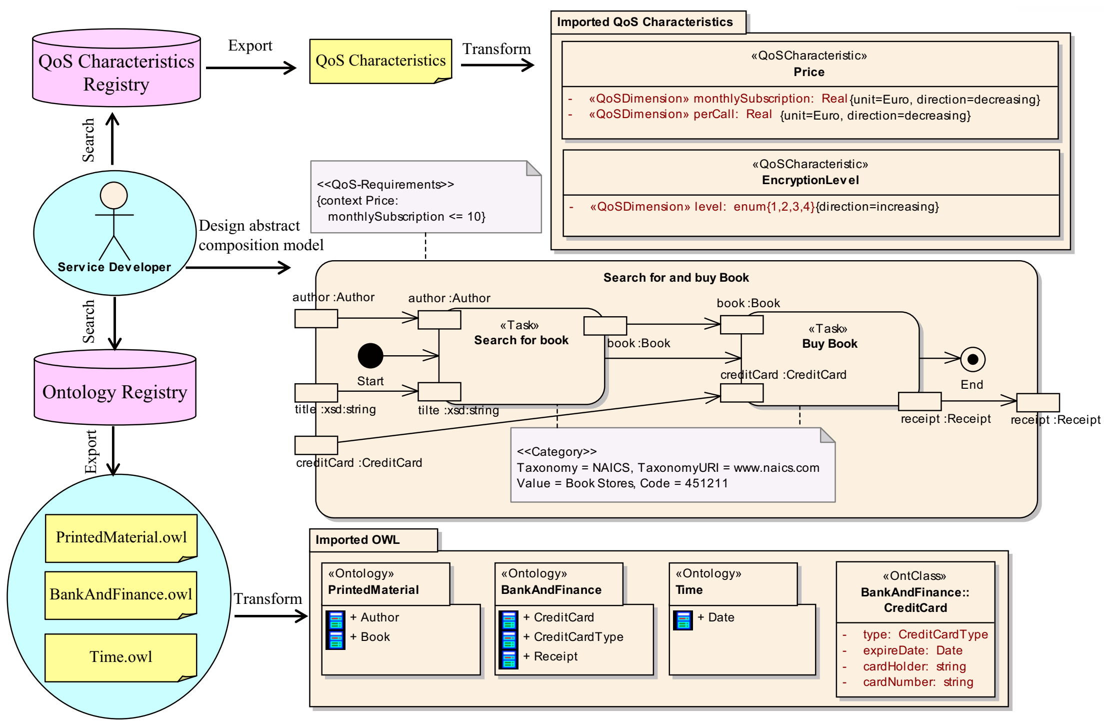

QoS Characteristics Registry
Export
QoS Characteristics
Transform
Imported QoS Characteristics
«QoSCharacteristic» Price
- «QoSDimension» monthlySubscription: Real{unit=Euro, direction=decreasing}
- «QoSDimension» perCall: Real {unit=Euro, direction=decreasing}
«QoSCharacteristic» EncryptionLevel
- «QoSDimension» level: enum{1,2,3,4}{direction=increasing}
Search
<<QoS-Requirements>> {context Price: monthlySubscription <= 10}
Service Developer
Design abstract composition model
Search for and buy Book
author :Author
author :Author
«Task» Search for book
Start
title :xsd:string
tilte :xsd:string
book :Book
book :Book
«Task» Buy Book
creditCard :CreditCard
End
receipt :Receipt
receipt :Receipt
creditCard :CreditCard
<<Category>> Taxonomy = NAICS, TaxonomyURI = www.naics.com Value = Book Stores, Code = 451211
Search
Ontology Registry
Export
PrintedMaterial.owl
BankAndFinance.owl
Time.owl
Transform
Imported OWL
«Ontology» PrintedMaterial
+ Author
+ Book
«Ontology» BankAndFinance
+ CreditCard
+ CreditCardType
+ Receipt
«Ontology» Time
+ Date
«OntClass» BankAndFinance:: CreditCard
- type: CreditCardType
- expireDate: Date
- cardHolder: string
- cardNumber: string

Figure 1. Abstract Composition Model with Tasks (phase one)

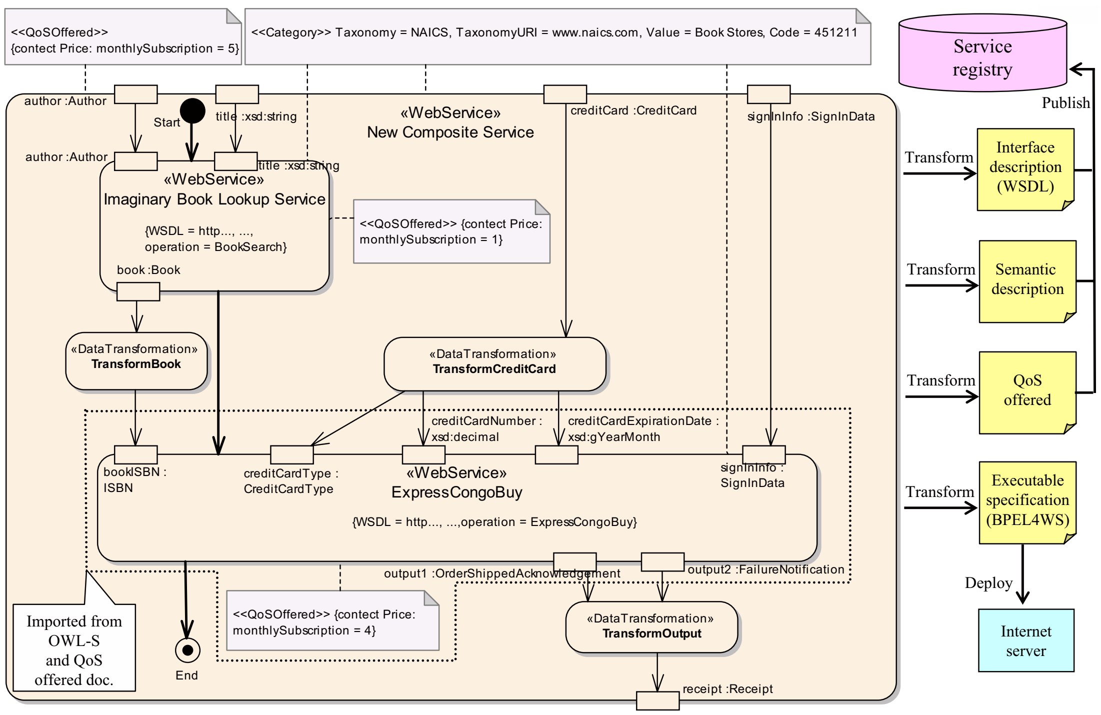
b) a semantic description that can be used for the matchmaking process. The abstract composition model (Figure 1) produced in the first phase is automatically transformed into a lexical document that can be parsed by a search and matchmaking algorithm.

The generated documents represent semantic descriptions of requested tasks for which we need to identify candidate services. We propose a matchmaking that deals only with semantic matching of inputs, outputs and categories [12]. However, even if a matching algorithm identifies clear matches for all three aspects we cannot guarantee the perfect match. For this aspect, further reasoning would be necessary that also take preconditions, postconditions and effects into account. Since this topic requires a lot of research, tools and adoption of such techniques by the user community, we do not expect that all services can be discovered fully automaticly in the near future. The matchmaking of inputs, outputs and categories will improve the precision of the search, but it will not remove the need for manual investigation of the discovered Web services to determine if they are suitable or not.

We anticipate that the found candidates provide a semantic description of their capabilities. The leading proposals for such semantic descriptions are OWL-S and WSML. However, the low-level and verbose OWL-S and WSML files are time-consuming and demanding to comprehend for developers. To ease the manual investigation process, we propose to *reverse engineer* OWL-S and WSML into high-level UML models. Another benefit of importing the semantic description into UML is that the imported services can be used directly as UML elements when finalizing the composition model. Based on investigation of the services in the graphical model, the user selects the appropriate services and ideally at least one chosen service is assigned to each task. This list of candidate services per task is the outcome of this phase.

4 Phases 3 and 4: Selection and Deployment

In this phase, we narrow down the set of Web services and rank the services based on the QoS requirements. Adopting the strategy in our preceding paper [6], the QoS requirements contain two parts: the first part covers absolute QoS constraints on the composition, the second part covers optimisation criteria that we use to rank the services. In preceding work, it is explained how to aggregate the QoS based on individual services [11] and how to apply selection algorithms that optimize the QoS of the compositions based on QoS categories as selection criteria [6]. The QoS-based sort and selection will return an assignment of a ranked list



<<QoSOffered>> {contect Price: monthlySubscription = 5}
<<Category>> Taxonomy = NAICS, TaxonomyURI = www.naics.com, Value = Book Stores, Code = 451211
Service registry
Publish
author :Author
Start
title :xsd:string
«WebService» New Composite Service
creditCard :CreditCard
signInInfo :SignInData
author :Author
title :xsd:string
«WebService» Imaginary Book Lookup Service
{WSDL = http..., ..., operation = BookSearch}
book :Book
<<QoSOffered>> {contect Price: monthlySubscription = 1}
«DataTransformation» TransformBook
«DataTransformation» TransformCreditCard
creditCardNumber : xsd:decimal
creditCardExpirationDate : xsd:gYearMonth
bookISBN : ISBN
creditCardType : CreditCardType
«WebService» ExpressCongoBuy
signInInfo : SignInData
{WSDL = http..., ...,operation = ExpressCongoBuy}
output1 :OrderShippedAcknowledgement
output2 :FailureNotification
Imported from OWL-S and QoS offered doc.
<<QoSOffered>> {contect Price: monthlySubscription = 4}
End
«DataTransformation» TransformOutput
receipt :Receipt
Transform
Interface description (WSDL)
Transform
Semantic description
Transform
QoS offered
Transform
Executable specification (BPEL4WS)
Deploy
Internet server

Figure 2. Concrete Composition Model with Chosen Web Services (phase three and four)

of candidate services for each task.

We assume that at least one realizing service is found for each task. Figure 2 shows a concrete composition model with one service for the two abstract tasks in the example taken over from Figure 1. We presume the existence of tools that perform automated reverse engineering from the semantic description and QoS offered documents into the graphical model. With these transformations, we can directly insert the imported Web services in place of their corresponding tasks in the composition model. Our goal is to fully automate this step. The example shows how we can import and reuse the semantically described Web service ExpressCongoBuy [2] into the composition model.

Each Web service is identified by a set of tagged values that uniquely identifies an operation inside a WSDL file. This is sufficient to be able to invoke the service operation as shown by Skogan et al. [19]. The expected or promised QoS offerings of each concrete service can be expressed with a QoS-offered stereotyped note attached to the activity. The category description of each service can be attached to the activity with a note stereotyped Category. The category description in our example is attached to both the atomic services and the composition itself since the same definition applies to all three.

The data objects and their types can be represented in the exact same manner as in the abstract model. We may expect that the outputs of one Web service does not perfectly match the required input of the next service. In such cases there is a need to introduce intermediate data transformation steps between the services. This step requires adjustments by the developer although there can be tools that assist the process. The details of the three needed data transformations in the example are not shown in the figure. In our example they only deal with copying the relevant parts of the transformation input to the relevant parts of the transformation output. The new composite service needs to be handled as a Web service itself and thus we need to assign explicit input/output parameters, category and QoS offerings to this composite activity. Compared to the abstract model, one input (signInInfo) was added to the composite service since this information is required by the discovered ExpressCongoBuy service. The outcome of phase three is a finalized concrete composition model.

In the fourth and last phase, the concrete composition model is used to generate different descriptions about the composed service: a) a WSDL document describing the syntactic interface and its technical bindings, b) a description about the offered and aggregated QoS, c) a semantic

[2]The Congo example used can be found at the OWL-S home page: http://www.daml.org/services/owl-s/1.1/examples.html.

description of the composed service, and d) an execution specification to execute the composition by an execution engine. The three generated items ($a - c$) are submitted to a Web service registry. Then, third parties can discover and use the composed service.

5 Additional Aspects of the Methodology

Our methodology identifies eleven model transformation steps that can be automated (Table 1). For each transformation we specify to which phase it belongs, the source and target in the transformation, and if there exist any proof-of-concept transformations or tools described in the literature. We have referenced only one proof-of-concept approach for each needed transformation even though there exists several proposals for a few of the transformations. The included proof-of-concept reference is not evaluated to be better or worse than the existing alternative proposals. We have mentioned UML in this context, however other graphical languages could be used as well. It is outside the scope of this paper to recommend a graphical language that best supports our methodology. There are five of the desired transformations for which we have not yet discovered any prototype tools. The same conclusion holds if we look beyond UML as the graphical language. The candidate lexical languages are either de-facto industry standards or promising proposals. We consider WSDL and BPEL4WS as the only de-facto industry standards, while the other lexical languages in the table are emerging languages and proposals that have the potential to be used within our methodology. The Unified Service Query Language (USQL [23]) can be used to express semantically annotated tasks for which one can perform searches for matching services.

Like almost all design and creation processes in the engineering and computer sciences, the design of the composition is an iterative and heuristic process that presumes to go back and forth to the different tasks of our methodology. Different scenarios which result in an iterative and heuristic methodology, are presented in the subsequent paragraphs.

No Matched Services. The discovery process and matching does not result in suitable service candidates. In this case the developer has two alternatives: either to implement the required service(s) himself, or to modify the composition design in order to find services that were previously excluded. In terms of our methodology, this means returning to the first phase to modify the composition.

Too Restrictive QoS. This may prevent the successful selection of services. Either because the matched candidates cannot satisfy the QoS requirements or because none of them provide information about the desired QoS category. In the first case, loosening up the QoS requirements is necessary. In the second case, the developer might consider to drop the previously defined constraint. In both cases, the developer needs to return to the first phase to modify the QoS requirements.

Too Many Candidates. As opposed to the previous two scenarios, a too large selection of suitable services might be found. In this case, the developer may define stricter QoS requirements and thereby optimise the service composition.

Methodology as Inspiration. During the matchmaking and reengineering process, the composition developer might discover that his abstract composition does not reflect an optimal design. Most likely, the developer will discover this when assessing the set of suitable service candidates. Then the process starts with the first phase again.

These issues show that the developer will continuously need to optimise the composition in order to approach the optimal setup by several points of possible iterations. Another reason to iterate is due to the continous change of available services. Some services become unavailable, while others are new or have changed QoS offerings since the last iteration.

6 Discussion

This section discusses if our methodology satisfies the requirements given in the Introduction section. The four requirements are discussed in the four subsequent paragraphs.

Models as the primary artefact. Our methodology satisfies the *Models as the primary artefact*-requirement since it enables the service developer to work only at the model level. All the existing specifications and relevant documents can be imported into models by reverse engineering tools. The manually updated abstract models can be used to generate all the lexical specifications needed to search and select services. In addition there are tools that shall enable the semi-automatically produced concrete models to automatically generate specifications needed to deploy and to publish information about the new service.

Automation. Our methodology has automated large parts of the steps needed in the process of developing composite services by identifying elleven model transformations. This implies that our methodology to a large extent satisfies the *automation* requirement.

Semantic support. The methodology proposes to semantically annotate (input/output types, category, precondition, postcondition, effect etc.) the atomic and composite Web services instead of lexical documents. This allows the service developer to completely define the new semantically annotated Web services graphically and then generate the lexical documents according to the semantic Web service languages. All the above-mentioned support are common for both OWL-S and WSML. WSML consists of *Goals* and *Mediators* in addition. The concept of *Goals* is supported in the methodology since it is a candidate for the lexical document being generated from the abstract model. *Mediators*

Table 1. Overview of automated transformations

Phase	Transformation	Source	Target	Proof-of-concept
1	Import domain ontologies	OWL, WSML Ontology	UML	OWL-to-UML [5]
1	Import QoS characteristics	WS-QoS [20]	UML	-
2	Export semantically annotated tasks	UML	WSML Goals, USQL [23]	-
2	Import semantic Web service	OWL-S, WSML WebService	UML	OWL-S-to-UML [7]
3	Export QoS requirements	UML	WS-QoS	-
3	Import QoS offered	WS-QoS, WSOL [22]	UML	-
3	Import Web service interface	WSDL	UML	WSDL-to-UML [8]
4	Export Web service interface	UML	WSDL	UML-to-WSDL [8]
4	Export semantic Web service	UML	OWL-S, WSML WebService	UML-to-OWL-S [21]
4	Export QoS offered	UML	WS-QoS, WSOL	-
4	Export executable specification	UML	BPEL4WS	UML-to-BPEL4WS [19]

are supported by the data transformations in our methodology, while the other kinds of *Mediators* are not supported. We conclude that our methodology satisfies the *semantic support* requirement to a large extent.

QoS support. The methodology utilizes QoS requirements in order to do a QoS-based sort and selection in phase 3. The automated component performing this task should also be capable of computing the aggregated QoS value for the new composite service, which in phase 4 is used to generate the QoS offerings document for the new service. We have chosen to perform the semantic discovery in phase 2 independently of the QoS selection in phase 3. As a result, the QoS selection considers a selected and narrowed set of service candidates and thus it will perform much quicker. By adopting the OMG's QoS profile we have a standard for representing the relevant QoS information in UML. This OMG profile could also be adjusted to be used as extensions to other graphical languages. The *QoS support* requirement is satisfied, but it is also the part of the methodology which is furthest from being realized by proof-of-concept tools.

7 Related Work

We have illustrated how the UML can be used as a graphical language in our methodology. Within model-driven service composition there are a few other promising proposals including Petri Net Models [9] and the Business Process Modelling Notation (BPMN) [1]. It is outside the scope of this paper to evaluate the suitability of the graphical languages for our methodology. However, for their application these languages would have have needed extensions to express the semantics and QoS. The OMG profile for modelling QoS [14] is a good starting point for UML as we have shown in our examples.

There are several proposals for a semantic Web service language including OWL-S [4], WSML[26] and WSDL-S [17]. Some of these have specialized graphical languages and tools to support the development in the respective languages such as the OWL-S Editor [2] and the WSMO studio [13]. Since there are multiple candidates for semantic Web service languages and these have not reached a finalized state, we recommend to use an independent graphical language. This would allow the graphical language to be an integration platform for Web services described using multiple semantic Web service languages.

The Web Service Offerings Language (WSOL) [22] and WS-QoS [20] represents an XML language for expressing QoS associated with Web services. They are both designed to be complementary with WSDL and they can both express QoS characteristics and offerings. WS-QoS can in addition express QoS requirements. As a graphical counterpart, the OMG profile for modelling QoS that can express all three QoS aspects that we need in our methodology (characteristics, requirements and offerings) [14]. We have pointed out the need to use a graphical language with defined transformations to the lexical QoS languages.

Several proposals exist to perform transformations between graphical languages and the semantic Web service languages including OWL-to-UML [5], UML-to-OWL-S [21] and OWL-S-to-UML [7]. There are however few transformations defined between other graphical languages and the semantic Web service languages.

8 Conclusions

This paper presents a model-driven methodology for designing composite Web services. The methodology considers a syntactic and semantic description about the interfaces of service candidates. It also processes QoS requirements from the developer and offerings from the service providers. In addition, we have identified which parts we can automate and which require an interactive handling by the developer. Many of the proposed steps for automation are model-driven transformations that transform between models and lexical descriptions about the Web services, both forward and reverse engineering. As the resulting benefit, the methodology provides a better documentation of the composition in the form of graphical models. The ability to generate semantically described and executable compositions from a graphical model represents a valuable gain to the service developers, who otherwise have to write a lot of low-level XML code.

Our approach is independent of the lexical languages for representing semantics and QoS of services. We assume that service developers will anticipate the flexibility to cover different semantic description languages and QoS representations. The methodology covers all the steps from the core idea of building a composite service until its final deployment and publication. Furthermore, we have identified how we can use models as the primary artefact by defining the characteristics of the required models and corresponding model transformations.

The methodology covers a lot of aspects which we can not investigate fully in a single paper. There are several issues that need more work to prove that our methodology is valid. There is a need to test the emerging lexical proposals for semantically annotated tasks (WSML Goals, USQL, etc.) to see whether they provide the anticipated level of expressive power to perform an efficient search and matchmaking of service offerings. Finally, we need to investigate further the suitable graphical service composition languages and accompanying tool-supported transformations to and from the lexical Web service documents.

Acknowledgments. The work of Roy Grønmo is partially funded by the European IST-FP6-004559 project SODIUM (Service Oriented Development In a Unified framework).

References

[1] BPMI.org. Business Process Modeling Notation (BPMN) Version 1.0, May 2004.

[2] Daniel Elenius et al. The OWL-S Editor - A Development Tool for Semantic Web Services. In *Proceedings of the 2nd European Semantic Web Conference (ESWC'05)*, Heraklion, Crete, Greece, May 2005.

[3] Daniela Berardi et al. Automatic Composition of Web Services in Colombo. In *Proceedings of the 13th Italian Symposium on Advanced Database Systems (SEBD'05)*, Brixen-Bressanone, Alto-Adige, June 2005.

[4] David L. Martin et al. Bringing Semantics to Web Services: The OWL-S Approach. In *Revised Selected Papers of the Intl Workshop Semantic Web Services and Web Process Composition (SWSWPC'04)*, San Diego, California, USA.

[5] D. Djuric. MDA-based Ontology Infrastructure. *Computer Science Information Systems (ComSIS)*, 1(1), February 2004.

[6] R. Grønmo and M. C. Jaeger. Model-Driven Methodology for Building QoS-Optimised Web Service Compositions. In *The 5th IFIP Intl Conference on Distributed Applications and Interoperable Systems (DAIS 2005)*, Athens, Greece, June 2005.

[7] R. Grønmo, M. C. Jaeger, and H. Hoff. Transformations between UML and OWL-S. In *European Conference on Model Driven Architecture – Foundations and Applications (ECMDA'05)*, Nuremberg, Germany, November 2005. (accepted for publication).

[8] R. Grønmo, D. Skogan, I. Solheim, and J. Oldevik. Model-Driven Web Service Development. *Intl Journal of Web Services Research (JWSR)*, 1(4), Oct-Dec 2004.

[9] R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. In *CRPITS'17: Proceedings of the Fourteenth Australasian database conference on Database technologies 2003*, Adelaide, Australia, 2003.

[10] ISO/IEC. ITU.TS Recommendation X.950 — ISO/IEC 13235-1: Trading Function: Specification, August 1997.

[11] M. C. Jaeger, G. Rojec-Goldmann, and G. Mühl. QoS Aggregation for Service Composition using Workflow Patterns. In *Proceedings of the 8th Intl Enterprise Distributed Object Computing Conference (EDOC'04)*, Monterey, California, September 2004.

[12] M. C. Jaeger, G. Rojec-Goldmann, G. Mühl, C. Liebetruth, and K. Geihs. Ranked Matching for Service Descriptions using OWL-S. In *Kommunikation in Verteilten Systemen (KiVS'05)*, Kaiserslautern, Germany, February 2005.

[13] H. Lausen and M. Felderer. Architecture for an Ontology and Web Service Modelling Studio. In *Proceedings of the Workshop on WSMO Implementations*, Frankfurt, Germany, September 2004.

[14] OMG. UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms, OMG Draft Adopted Specification. OMG Document: ptc/2004-06-01, June 2004.

[15] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. In *Revised Papers from the Intl Workshop on Web Services, E-Business, and the Semantic Web*, Toronto, Canada, May.

[16] J. Peer. A PDDL Based Tool for Automatic Web Service Composition. In *Proceedings of the Second Intl Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR)*, St. Malo, France, September 2004.

[17] P. Rajasekaran, J. A. Miller, K. Verma, and A. P. Sheth. Enhancing Web Services Description and Discovery to Facilitate Composition. In *Revised Selected Papers of the Intl Workshop Semantic Web Services and Web Process Composition (SWSWPC'04)*, San Diego, California, USA.

[18] S. Ran. A Model for Web Services Discovery with QoS. *SIGecom Exch.*, 4(1), 2003.

[19] D. Skogan, R. Grønmo, and I. Solheim. Web Service Composition in UML. In *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC'04)*, Monterey, California, September 2004.

[20] M. Tian, A. Gramm, H. Ritter, and J. Schiller. Efficient Selection and Monitoring of QoS-Aware Web Services with the WS-QoS Framework. In *Proc. of the IEEE/WIC/ACM Intl Conference on Web Intelligence (WI'04)*, Washington, DC, USA, September 2004.

[21] J. T. E. Timm and G. C. Gannod. A Model-Driven Approach for Specifying Semantic Web Services. In *Proceedings of the IEEE Intl Conference on Web Services (ICWS'05)*, Orlando, Florida, USA, July 2005.

[22] V. Tosic, K. Patel, and B. Pagurek. WSOL - Web Service Offerings Language. In *Revised Papers from the Intl Workshop on Web Services, E-Business, and the Semantic Web (CAiSE'02/WES'02)*, 2002.

[23] A. Tsalgatidou, M. Pantazoglou, and G. Athanasopoulos. SODIUM deliverable D8 - Specification of the Unified Service Query Language (USQL) v.0.4, June 2005.

[24] UDDI Spec Technical Committee. UDDI Version 3.0.1. http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf, 2003.

[25] W3C. Web Services Architecture W3C Working Group Note, February 2004. http://www.w3c.org/TR/ws-arch/.

[26] WSMO Working Group. D16.1v0.2 The Web Service Modeling Language WSML, March 2005.