Semantic Model-Driven Development of Service-centric Software Architectures

Claus Pahl and Ronan Barrett

Dublin City University
School of Computing
Dublin 9, Ireland
[cpahl|rbarrett]@computing.dcu.ie

Abstract. Service-oriented architecture (SOA) is a recent architectural paradigm that has received much attention. The prevalent focus on platforms such as Web services, however, needs to be complemented by appropriate software engineering methods. We propose the model-driven development of service-centric software systems. We present in particular an investigation into the role of enriched semantic modelling for a model-driven development framework for service-centric software systems. Ontologies as the foundations of semantic modelling and its enhancement through architectural pattern modelling are at the core of the proposed approach. We introduce foundations and discuss the benefits and also the challenges in this context.

Keywords: Service-oriented Architecture, Service Processes, Model-Driven Development, Semantic Modelling, Ontologies, Architectural Patterns.

1 Introduction

Service-oriented architecture (SOA) has been successfully deployed as an architectural paradigm in recent years [2]. The dominance of the Web services platform in distributed systems development and deployment has given a boost to SOA as an approach to software architecture [3]. Current solutions to description and collaboration languages and to platform technologies such as protocols need to be embedded into an adequate software engineering methodology.

The SOA approach creates some specific requirements for supporting software engineering methods [23]. Firstly, services are used 'as is', with providers and consumers often coming from different organisations. This requires detailed descriptions of functional and non-functional aspects in an mutually agreed format Secondly, service descriptions need to be based on rich modelling languages, in order to support their automated discovery and composition. Thirdly, reuse is a central aim of service technology that needs to be addressed on the level of individual services as well as service compositions.

Model-driven development (MDD) is an established and widely used approach to software development that aims at cost-effective automation and improved maintainability through abstract modelling. The recent standardisation

of the approach, Model-Driven Architecture (MDA), has emphasised the importance of this approach for the software sector [1]. MDA proposes abstract modelling combined with the automatic generation of code from these models.

We propose an MDD approach for SOA, aiming to support the development of service-centric software systems. We will explore foundations (in form of ontologies) and design methods (in form of architectural patterns). Specifically, we deem two specific methods necessary to satisfy the specific requirements:

- semantic modelling supported through ontologies, i.e. logic-based knowledge representation frameworks, in order to support rich modelling constructs, reasoning, and formal semantics [15].
- pattern-based modelling supported through architectural patterns in order to add higher levels of abstraction and to enable the reuse of useful architectural styles and designs.

Our contribution is twofold. Firstly, a discussion of central techniques for MDD for service-centric software systems. Secondly, an outline of an ontology-based framework for MDD of service-centric software systems involving architectural patterns as central design technique – these are patterns (or styles) of architectures, typically different from design patterns [24]. Our investigation shall lead towards an emerging service engineering discipline.

We divide our discussion into three aspects. We start with modelling of service functionality supported by ontologies in Section 2. Modelling of non-functional aspects and the use of patterns is then the topic of Section 3. We discuss ontologies and patterns in a wider context of service engineering in Section 4. We end with a discussion of related work and some conclusions.

2 Modelling of Service Functionality

Modelling of software often focuses on functional aspects such as the software architecture or behavioural aspects of a software system. This traditional focus of software modelling shall also be discussed in this section and the specific modelling needs for service-centric software systems shall be discussed.

Abstraction and automated code generation are the pillars of model-driven development [4]. We investigate the role that ontologies can play for the support of these pillars in the context of functional aspects of service-oriented architecture. In terms of the OMG's MDA framework, we are going to address modelling at the platform-independent (PIM) layer and transformations form there into the platform-specific layer (PSM).

2.1 Semantic Service Modelling

UML is the most widely used software modelling notation. An integration of the proposed semantic modelling approach with UML is important for two reasons:

- UML provides a rich combination of individual visual modelling languages. These visual interfaces can be adapted to support service modelling.

– A vast amount of UML models for a broad range of application contexts exist. Reusing and integrating these is almost a necessity.

UML is limited in particular in terms of the constraints on software systems that can be expressed. OCL, the Object Constraint Language, is a UML extension that allows semantic constraints such as pre- and postconditions and invariants for operations and classes to be expressed [17]. While a wide range of powerful logics have been available to support constraint formulation and reasoning, the recent advent of the Semantic Web with its ontology-based foundations have make ontologies a promising candidate for enhanced semantic modelling in the context of the Web platform [16].

– Ontologies provide a logical language, for instance based on description logic [14], with its reasoning support.
– Ontology languages, such as OWL, are the knowledge representation languages of the Web [16] that enable XML-based syntactical interoperability and knowledge sharing and exchange.

For these reasons, the semantic modelling of Web services is ideally supported by ontologies, as the research focus on semantic Web services shows [7, 12]. Besides providing a rich semantic modelling notation – which can, as proposed, be augmented by a UML-style interface – ontologies can support design activities such as composition, and act as formal semantics for the overall model-driven development integrating modelling and transformations.

WSMO [10] and OWL-S [9] are the two predominant examples of service ontologies. Service ontologies are ontologies to describe Web services, aiming to support their semantics-based discovery in Web service registries. The Web Service Process Ontology WSPO [13] is also a service ontology, but its focus is the support of description and reasoning about service composition and service-based architectural configuration. An important current development is the Semantic Web Services Framework (SWSF), consisting of a language and an underlying ontology [21], which takes OWL-S work further. The FLOWS ontology in SWSF comprises process modelling and is like WSPO suited to support semantic service modelling within the MDA context. While the early service ontologies have focused on individual services and their properties – OWL-S and WSMO – more recent attention has been paid to service process ontologies – SWSF and WSPO – which focus on the composition of services to form processes. A different type of ontology, supporting composition activities and the formulation of collaborating servcies as processes, is needed for the composition aspect [8].

2.2 Service Composition

In order to support the development of service architectures, more than just description notations are needed. Specific activities such as service discovery or process composition need to be supported through anaysis and reasoning techniques [22]. We look here into ontology-based support for service composition.

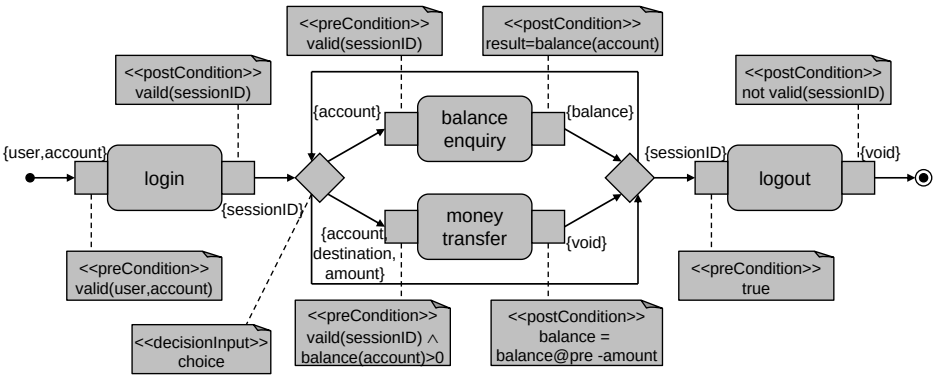

Fig. 1. Semantic Service Process Model based on UML Activity Diagrams.

Service composition is a logical composition where an abstract service process based on individual services is assembled [6]. We have developed the WSPO composition ontology – Web Services Process Ontology – in order to support the semantically correct composition of services [13].

Ontologies in general consist of concepts that represent objects, process, etc. and relationships between these concepts. Services (and processes) in the WSPO ontology are not represented as concepts, but as relationships denoting accessibility relations between states of a software system.

- Ontology concepts in this approach are states (pre- and poststates), parameters (in- and out-parameters), and conditions (pre- and postconditions).
- Two forms of relationships are provided in the ontology. The services or processes themselves are called transitional relationships. Syntactical and semantical descriptions here parameter objects (syntax) and conditions (semantics) are associated through descriptional relationships.

WSPO provides a standard template for service and service process description – applied in Fig. 1 to express a composite bank account process consisting of operations provided by services like 'balance enquiry' or 'money transfer'. This application is based on four individual services, each described in terms of input, output, precondition, and postcondition. Syntactical parameter information in relation to the individual activities – to be implemented through service operations – and also semantical information such as pre-conditions are attached to each activity as defined in the template. The following textual representation summarises the syntactical aspects in a traditional IDL-style interface format:

application AccountProcess
  services login (user:string, account:int) : ID
    balance enquiry (account:int) : currency
    money transfer (account:int, destination:int, amount:currency) : void

logout (sessionID:ID) : void
process login; !(balance enquiry + money transfer); logout

The services are composed to a process, here using the combinators sequence (;), iteration (!), and choice (+) in the textual representation above, which is also visualised in UML notation in Fig. 1.

WSPO can be distinguished from traditional service ontologies by two specific properties. Firstly, based on an extension of description logics [14], it adds a relationship-based process sublanguage enabling process expressions based on iteration, sequential and parallel composition, and choice operators. Secondly, it adds data to processes in form of parameters that are introduced as constant process elements into the process sublanguage. This ontological representation in WSPO is actually an encoding of a simple dynamic logic (a logic of programs) in a description logic format [13], allowing us to avail of modal logic reasoning about processes in this framework. For example, in order to implement a bank account process, an implementation for a `money transfer` service with input parameter `amount` (assumed to be always positive) needs to be integrated. For any given state, the process developer might require (using the `balance` enquiry)

```
service:preCondition  rdfConstr="balance() > amount"
service:postCondition rdfConstr="balance() = balance()@pre - amount"
```

which would be satisfied by a provided service

```
service:preCondition  rdfConstr="balance() > 0"
service:postCondition rdfConstr="balance() = balance()@pre - amount
    and lastActivity = 'transfer'"
```

The provided service would weaken the required precondition assuming that the transfer `amount` is always positive and strengthen the required postcondition as an additional result (`lastActivity`) is delivered by the provided service. This matching notion, which is important for service discovery support, is implemented in WSPO based on a refinement of the standard subsumption relation of description logics towards a consequence inference rule from dynamic logic.

We have chosen WSPO here instead of the SWSF. The more recent SWSF is an equally suitable service process ontology, providing actually a wider range of modelling feature, which would make it the better choice once sufficient tool support is available. For the moment, WSPO as a more focused and decidable ontology is the better choice in terms of tractability of the approach.

2.3 Transformation and Code Generation

Automated code generation is one of the central objectives of MDD. Two types of generation of code in service platform-specific languages are relevant here

– Executable code generation: in the context of SOA, code generation essentially means the generation of exectuable service processes. WS-BPEL

(BPEL, 2004) has emerged as the most widely accepted process execution language for Web services. Within MDD, code generation is usually specified through transformation rules.
– Abstract description generation: a central element of SOA is service discovery based on abstract service descriptions. OWL-S, WSMO, or SWSF would be suitable ontology frameworks that support description and discovery.

An overview of some of the transformation rules for a WSPO to WS-BPEL transformation to generate executable code is presented below. WSPO defines a simple process language that can be fully translated into WS-BPEL. BPEL process partners are the consumers and the different service providers – the WSPO specification is already partitioned accordingly. The WSPO process combinators can also be mapped directly onto the BPEL flow combinators. The principles of this transformation can be characterised as follows:

– The WSPO process relationships can be mapped to BPEL processes.
– For each process, a BPEL partner process (consumer and server) is created.
– Each process expression is converted into BPEL-invoke activities at the client side and BPEL-receive and -reply activities at the server side.
– The process combinators ';', '+', '!', and '||' are converted to the BPEL flow combinators sequence, pick, while, and flow, respectively.

A schematic example in the declarative transformation language QVT shall illustrate these principles for WSPO-to-BPEL transformations[1]:

```
transformation WSPO_to_WS-BPEL (wspo : WSPO, bpel: WS-BPEL)
{
  top relation TransRelationship_to_Process
                   /* maps trans. relationships to processes */
  {
    checkonly domain wspo w:TransRel {name = pn}
    enforce   domain bpel b:Process  {name = pn}
  }
  where {
      RelExpr_to_ProcessExpr(w);
  }

  relation RelExpr_to_ProcessExpr
                   /* map each process expression recursively */
  {
    domain wspo r:Relationship {
      e1 = e:LeftExpr {},
      e2 = e:RightExpr {},
      op = c:Combinator {}
```

[1] Although its standardisation is not completed and no tool support is currently available for the declarative specifications, QVT will be an essential tool for the implementation of our approach in the near future.

```
    }
    domain bpel p:Process {
      process = p:Process {left=e1, pr=op, right=e2},
      client = pe:ProcessExpr {invoke=op(e1,e2)},
      server = pe:ProcessExpr {receive=(op,e1,e2), reply=op(e1,e2)}
    }
    when {
      CombinatorMapping(op) and ProcessPartnerCreation(p);
    }
    where {
      RelExpr_to_ProcessExpr(e1) and RelExpr_to_ProcessExpr(e2);
    }
  }
}
```

We do not present the WS-BPEL representation here, since with the exception of some reformulations of operators and the additional verbosity of the XML-based formulation, the process itself is similar to the WSPO representation.

The second transformation and code generation context relates to service description and discovery. The Web Services platform proposes a specific architecture based on services, which can be located using abstract service information provided in service registries. The description of services – or service processes made available as a single service - ideally in semantical format is therefore of central importance. Information represented in the process model and formalised in the service process ontology can be mapped to a service ontology. This transformation would only be a mapping into a subset of these ontologies, since they capture a wide range of functional and non-functional properties, whereas we have focussed on architecture-specific properties in WSPO.

We chose WSMO as the target. An overview of the transformation rules from WSPO to WSMO is outlined below. Some correspondences between the two ontology frameworks guide this transformation. WSPO input and output elements correspond to WSMO message-exchange patterns, which are used in WSMO to express stimuli-response patterns of direct service invocations, and WSPO pre- and postconditions correspond to their WSMO counterparts.

- WSPO process relationships are mapped to WSMO service concept and fill messageExchange and pre- and postCondition properties accordingly.
- WSPO in/out-objects are mapped to WSMO messageExchange descriptions.
- WSPO pre- and postconditions are mapped onto their WSMO counterparts.

A QVT formulation would be similar to the WSPO-to-BPEL transformation.

2.4 Discussion

This section has demonstrated the role that ontologies can play in model-driven development of service-centric software systems. They can provide a semantic modelling framework in particular for the Web platform due to the support of

ontologies through the Semantic Web initiative. They can act as a foundational layer to define visual modelling languages, to support composition tasks, and to enable automated transformations.

While we have discussed the application of ontologies for functional aspects of software systems specification such as pre- and postconditions, their scope stretches further to also include non-functional aspects such as general descriptions, cost models, security requirememts, etc. . Non-functional aspects shall be addressed in depth in the next section.

3 Modelling of Non-Functional Service Aspects

The deployment model of services in general and Web services in particular is based on the idea of service provider and service consumer being business partners. This constellation requires contracts to be set up, based on service-level agreements (SLAs). While of course functional characteristic of services are vital elements in these SLAs, non-functional aspects – ranging from availability and performance guarantees to costing aspects – are equally important and need to be captured in SLAs to clearly state the quality-of-service (QoS) obligations and expectations of provider and consumer.

This discussion will show that ontologies are an adequate modelling notation, but that additional techniques are needed to address modelling, in particular if functional and non-functional aspects are integrated. We will introduce distribution patterns, again at the architectural level, to provide a framework for higher levels of abstraction beyond the service process composition [26]. These patterns will add a widely used method centering around the notion of reuse of abstract architectural designs and models to semantic modelling. A quality dimension is added through quality attributes associated to these distribution patterns.

3.1 Distribution and Service Topology

Ontologies, the focus of the previous section, can deal with the abstract specification of QoS properties by providing an adequate vocabulary based on a variety of relationships. A mere statement of required QoS properties is therefore often not sufficient to actually guarantee these properties. However, there are links between functionally-oriented models and QoS properties. We look at distribution properties of service-centric software systems to illustrate this.

Distribution, i.e. the consideration of the location of services in a complex system, affects qualities of the software systems such as reliability, availability, and performance. We use the term service topology to refer to the modelling of service compositions as collaborating entities under explicit consideration of the distribution characteristics.

3.2 Service Topology Modelling

Based on experience with the design and implementation of service-centric software systems, a number of standard architectural configurations have emerged

[25, 27, 29, 31]. These include centralised configurations such as the Hub-and-Spoke or decentralised ones such as Peer-to-Peer architectures. These standard configurations can be abstracted into architecture-lavel distribution patterns for the SOA platform.

The goal is to enable the generation of architecturally flexible Web service compositions, whose Quality of Service (QoS) characteristics can be evaluated and altered at design time. Distribution pattern modelling expresses how the composed system is to be deployed from the architectural perspective [18]. Having the ability to model, and thus alter the distribution pattern, allows an enterprise to configure its systems as they evolve, and to meet varying non-functional requirements.

There is a subtle difference between two of the modeling aspects within a Web service composition, namely workflows and distribution patterns. Both aspects refer to the high level cooperation of components, termed a collaboration, to achieve some compound novel task. We consider workflows as compositional orchestrations, whereby the internal and external messages to and from services are modelled. In contrast, distribution patterns are considered compositional choreographies, where only the external messages flow between services is modelled. Consequently the control flow between services are considered orthogonal. As such, a choreography can express how a system would be deployed. The internal workflows of these services are not modelled here, as there are many approaches to modelling the internals of such services.

3.3 Modelling Process and Transformation

This component of our framework comprises a catalog of distribution patterns, which may be applied by software architects to Web service compositions [18]. Distribution patterns express how a composed system is to be assembled and subsequently deployed. Each of the patterns in the catalog has associated Quality of Service (QoS) characteristics, exhibited during execution of Web service compositions. The catalog enumerates the QoS characteristics of each of the patterns, enabling the architect to choose a pattern appropriate to the non functional requirements of a given composition.

The patterns are split into three categories, core patterns, auxiliary patterns and complex patterns. Core patterns represent the simplest distribution patterns most commonly observed in Web service compositions. Auxiliary patterns are patterns which can be combined with core patterns to improve a given QoS characteristic of a core pattern, the resultant pattern is a complex pattern. This catalog assists software architects in choosing a distribution pattern for a given application context. The catalog is outlined briefly below:

- Core Patterns: Centralised, Decentralised
- Auxiliary Patterns: Ring
- Complex Patterns: Hierarchical, Ring + Centralised, Centralised + Decentralised, Ring + Decentralised

We describe one pattern to illustrate distribution patterns and their QoS relevance. The Centralised pattern, or Hub-and-Spoke pattern, manages the composition from a single location, normally the participant initiating the composition. Here is the service choreography:

application Centralised
  services Hub (in : inType) : outType
  Spoke$_1$ (...) : ...
  ...
  Spoke$_n$ (...) : ...
  process Hub $||_D$ (Spoke$_1$ || ... || Spoke$_n$)

The composition controller (the hub) is located externally from the service participants to be composed (the spokes), indicated through the distribution annotation at the parallel composition operator. Spokes would internally invoke the Hub using the given Hub interface. The external spoke interfaces are not relevant here. This is the most popular and usually default distribution configuration for service compositions. The advantages of the pattern in terms of QoS aspects are:

- Composition is easily maintainable, as composition logic is all contained at a single participant, the central hub.
- Low deployment overhead as only the hub manages the composition.
- Composition can consume participant services that are externally controlled. Web service technology enables the reuse of existing services.
- The spokes require no modifications to take part in the composition. Web service technology enables interoperability.

The main disadvantages are:

- A single point of failure at the hub provides for poor reliability/availability.
- A communication bottleneck at the hub restricts scalability. SOAP messages have considerable overhead for message deserialisation and serialisation.
- The high number of messages between hub and spokes is sub-optimal. SOAP messages are often verbose resulting in poor performance for Web services.
- Poor autonomy in that the input and output values of each participant can be read by the central hub.

A distribution pattern language DPL provides the constructs for the internal representation of a distribution pattern. The DPL, which similarly to WSPO has a UML interface based on activity diagrams, provides:

- control flow: information about the nodes and edges that define the control flow structure, on which interactions between individual services take place.
- data flow: information about which data is flowing in which direction in an interaction between services.

DPL is a high-level service process modelling language that provides the foundations for pattern-based service architecture.

3.4 Discussion

While we have looked at architectural patterns to model the functional side in the previous section, i.e. the structural and behavioural aspects of software, we have focused here on their potential in the context of non-functional properties. Distribution patterns imply non-functional properties of service-centric software systems. The choice of these patterns in particular determines performance, availability, and reliability charateristics of the software system itself.

4 A Service Engineering Perspective

Issues arising from the cooperation between organisations and the integration of systems across organisations leads to another couple of issues relevant to our discussion of ontologies and patterns for service-centric software engineering. This discussion of a wider context helps us to determine the potential, but also the challenges for the use of ontologies and patterns.

4.1 Cooperation

The cooperation between service providers and consumers is necessary for service-oriented computing. This requires at least interoperability of formats of all artifacts involved, in particular for models. Contracts stating service-level agreements are the actual documents, which might refer to additional documents and models internally. Ontologies have the potential to provide a common vocabulary that would allow the automated processing of these contracts. The wider aim within a service-centric software engineering discipline is an integrated value chain for software services that brings together all participants in the production process – for instance through interoperable deployment infrastructures, but also through contract and model exchange.

4.2 Trust

Trust is a central problem for collaborating partners that are initially often unknown to each other. In the context of automated service compositions, detailed model-based contracts can capture the obligations and expectations in terms of functional and QoS service properties, but do not suffice on their own. Interoperability is an enabler of integrated value chains, but only trust will make this business scenario work ultimately.

At the core of trust mechanisms are composition and description techniques, for instance our ontology-based modelling techniques. Quality-of-service and other non-functional properties broaden the overall focus without losing the composition activity at the core.

The certification of services is the central trust mechanism. A certification infrastructure needs to be added to allow in particular consumers to trust their service providers. Certification needs to ensure in the service composition context

more than the authenticity of the provider; it needs to ensure service properties. This is another potential use of, but also a technical challenge for ontology technology. Models of the various aspects play a central role here as the foundations of contracts, which can form the basis of a certificate. For instance, the proofs of properties resulting from ontological reasoning about service properties and service compositions can be included in these certificates.

Trust is an important non-functional consideration that effects the business context of service composition and provision. The richness of our ontological modelling framework in terms of models and reliable patterns, however, makes trust also an opportunity that might succeed for this context.

4.3 Environment and Standardisation

The activities by standardisation bodies in the software technology context, such as the OMG or the W3C, indicate two directions for model-driven service engineering. The OMG has proposed MDA to address code generation and maintenance, which can act as a framework for a development approach for service-centric software systems.

The integration of models and descriptions is the first challenge. One standard is expected to be central in this endeavour. The Ontology Definition Metamodel ODM is a MOF-based integration format for ontologies and UML models, currently standardised by the OMG [11]. The ODM metamodel can support semantic services, process composition, and UML model reuse. It allows the overall integration of semantic descriptions and models. ODM is complemented by the W3C's proposal of OWL as the Semantic Web ontology language and ODA as an ontology-based software development approach.

A second challenge arises from the business perspective on services. This would include in the services context in particular workflow and business process modelling as part of a business process engineering stage and semantic integration. Currently, this aspect with a focus on business modelling and analysis is receiving some attention. The OMG-supported Business Process Modelling Notation BPMN is an example. MDA, for instance, provides only a framework, but not enough support for the consistent mapping of business processes onto the platform layer. The use of ontologies for semantic integration and patterns to provide an abstraction and design technique across the stages would improve this endeavour.

5 Related Work

Some approaches have started exploiting the connection between ontologies – in particular OWL – and MDA. In [19], an MDA-based ontology architecture is defined. This architecture includes aspects of an ontology metamodel and a UML profile for ontologies. The work by [19] and the OMG [1, 11], however, needs to be carried further to address the ontology-based modelling and reasoning of

service-based architectures. In particular, the Web Services architecture needs to be addressed in the context of Web-based ontology technology.

Grønmo et.al. [20] introduce – based on ideas from [19] – an approach similar to ours. Starting with a UML profile based on activity diagrams, services are modelled. These models are then translated into OWL-S. Although the paper discusses process composition, this aspect is not detailed. We have built on [20] in this respect by considering process compositions in the UML profile and by mapping into a service process ontology that focusses on providing explicit support for service composition. In comparison to UML/OCL approaches, ontologies enable full-scale logic support and also the possibilty to share representations due to their grounding in OWL.

Since software architecture is the overall context of our investigation, architectural description languages (ADLs) shall briefly be discussed. For instance, Darwin [28] is a π-calculus-based ADL. Darwin focuses on component-oriented development, addressing behaviour and interfaces. Restrictions based on the declarative nature of Darwin make it rather unsuitable for the design of service-based architectures, where both binding and unbinding on demand are required features. With the support for composition and abstraction of architectural configurations service composition ontologies and distribution patterns would constitute an essential part of an ADL.

The notion of patterns has recently been discussed in the context of Web service architectures [25, 29]. In [29, 27], collections of workflow patterns are compiled. We have based our catalog on these collections. Grønmo et al. [30] consider the modelling and building of compositions from existing Web services using MDA, based on [31]. The authors consider two modelling aspects, service (interface and operations) and workflow models (control and data flow concerns). Their modelling effort begins with the transformation of WSDL documents to UML, followed by the creation of a workflow engine-independent UML activity diagram, which drives the generation of an executable composition.

6 Conclusions

A new architectural paradigm such as service-oriented architecture (SOA) requires adequate methodological support for design and maintenance. While an underlying deployment platform exists in the form of the Web services platform, an engineering methodology and techniques are still largely missing. The importance of modelling for SOA has been recognised – and has resulted in the development of Model-Driven Architecture (MDA) as an approach to support the design of service-centric software systems. We have focussed on pattern-based semantic modelling of service architectures. These are two architectural aspects – semantic service description and pattern- and process-based architectural configuration – that can complement the MDA approach.

Ontology and Semantic Web technologies provide semantic strength for the modelling framework, necessary for a distributed and inter-organisational environment. A central element of our modelling method is a service ontology

tailored to support service composition and transformation. An ontology-based technique is here beneficial for the following reasons. Firstly, ontologies define a rigourous, logic-based semantic modelling and reasoning framework thats support architectural design activities for services. Secondly, ontologies provides a knowledge integration and interoperability platform for multi-source semantic service-based software systems. Our aim here was to demonstrate the suitability of ontologies for this environment – for both WSPO to support architectural issues but also for WSMO here to support service discovery. We have integrated this service composition ontology with a pattern-based architecture modelling technique integrating visual UML-based modelling, transformation, ontology-based reasoning, and code generation.

We see our investigation as a step towards a service engineering discipline. Service engineering deals with process and integration issues. While we have focused on service composition and integration, data integration is an equally important that needs to be investigated further in this context. We interpret here a notion of service engineering as a wider process, covering a number of value chain stages. The software value chain that we have referred to earlier structures service engineering. It is a classical top-down model that needs to be augmented further. System evolution adds another dimension of importance for a service engineering discipline. Re-engineering and migration need to be integrated. Legacy systems can be integrated into service-oriented architectures using software re-engineering and architecture transformation.

References

1. Object Management Group. *MDA Model-Driven Architecture Guide V1.0.1.* OMG, 2003.
2. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd Edition).* SEI Series in Software Engineering. Addison-Wesley, 2003.
3. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications.* Springer-Verlag, 2004.
4. B. Selic. The Pragmatics of Model-Driven Development, *IEEE Software*, 20(5):19–25, 2003.
5. World Wide Web Consortium. *Web Services Architecture.* http://www.w3.org/TR/ws-arch, 2006. (visited 28/02/2006).
6. C. Peltz. Web Service orchestration and choreography: a look at WSCI and BPEL4WS. *Web Services Journal*, 3(7), 2003.
7. S. McIlraith and D. Martin. Bringing Semantics to Web Services. *IEEE Intelligent Systems*, 18(1):90–93, 2003.
8. J. Rao, P. Küngas, and M. Matskin. Logic-Based Web Services Composition: From Service Description to Process Model. In *International Conference on Web Services ICWS 2004*, pages 446–453. IEEE Press, 2004.
9. DAML-S Coalition. DAML-S: Web Services Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.
10. R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

11. Object Management Group. *Ontology Definition Metamodel - Request For Proposal (OMG Document: as/2003-03-40).* OMG, 2003.
12. C. Pahl. Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture. In *European Conference on Model-Driven Architecture ECMDA2005.* Springer LNCS Series, 2005.
13. C. Pahl. An Ontology for Software Component Matching. *International Journal on Software Tools for Technology Transfer (STTT), Special Edition on Component-based Systems Engineering*, 7, 2007 (in press).
14. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.
15. M.C. Daconta, L.J. Obrst, and K.T. Smith. *The Semantic Web.* Wiley, 2003.
16. W3C Semantic Web Activity. Semantic Web Activity Statement, 2004. http://www.w3.org/2001/sw.
17. J.B. Warmer and A.G. Kleppe. *The Object Constraint Language – Precise Modeling With UML.* Addison-Wesley, 2003. (2nd Edition).
18. R. Barrett, L. M. Patcas, J. Murphy, and C. Pahl. Model Driven Distribution Pattern Design for Dynamic Web Service Compositions. In *International Conference on Web Engineering ICWE06. Palo Alto, US.* ACM Press, 2006.
19. D. Djurić. MDA-based Ontology Infrastructure. *Computer Science and Information Systems (ComSIS)*, 1(1):91–116, 2004.
20. R. Grønmo, M.C. Jaeger, and H. Hoff. Transformations between UML and OWL-S. In A. Hartman and D. Kreische, editors, *Proc. Model-Driven Architecture – Foundations and Applications*, pages 269–283. Springer-Verlag, LNCS 3748, 2005.
21. Semantic Web Services Language (SWSL) Committee. *Semantic Web Services Framework (SWSF).* http://www.daml.org/services/swsf/1.0/, 2006.
22. B.-H. Schlingloff, A. Martens and K. Schmidt. Modeling and Model Checking Web Services. *Electronic Notes in Theoretical Computer Science: Issue on Logic and Communication in Multi-Agent Systems*, 126:3-26. 2005.
23. R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *Intl. Journal of Cooperative Information Systems*, 13(4):337-368. 2004.
24. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Design.* Addison Wesley, 1995.
25. N.Y. Topaloglu and R. Capilla. Modeling the Variability of Web Services from a Pattern Point of View. In L.J. Zhang and M. Jeckle, editors, *Proc. European Conf. on Web Services ECOWS'04*, pages 128–138. Springer-Verlag, LNCS 3250, 2004.
26. C. Pahl and R. Barrett. Towards a Re-engineering Method for Web Services Architectures. In *Proc. 3rd Nordic Conference on Web Services NCWS'04*. 2004.
27. W.M.P. van der Aalst, B. Kiepuszewski A.H.M. ter Hofstede, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.
28. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In W. Schäfer and P. Botella, editors, *Proc. 5th European Software Engineering Conf. (ESEC 95)*, Springer LNCS 989, pages 137–153. 1995.
29. M. Vasko and S. Duskar. An Analysis of Web Services Flow Patterns in Collaxa. In L.J. Zhang and M. Jeckle, editors, *Proc. European Conf. on Web Services ECOWS'04*, pages 1–14. Springer LNCS 3250, 2004.
30. D. Skogan, R. Grønmo and I. Solheim. Web Service Composition in UML. In *Proc. 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC), pages 4757.*2004.
31. S. Thöne, R. Depke and G. Engels. Process-Oriented, Flexible Composition of Web Services with UML. In *Proc. Joint Workshop on Conceptual Modeling Approaches for e-Business (eCOMO 2002).* 2002.