View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DCU Online Research Access Service

Ontology Support for Web Service Processes

C. Pahl
Dublin City University
School of Computing
Dublin 9, Ireland
cpahl@computing.dcu.ie

M. Casey
Dublin City University
School of Computing
Dublin 9, Ireland
mcasey@computing.dcu.ie

ABSTRACT

Web Services are software services that can be advertised by providers and deployed by customers using Web technologies. This concept is currently carried further to address Web service choreography. Choreography refers to the composition of individual services to services processes that can communicate and interact with another.

We propose a formal ontology framework for these Web service processes that supports the description, matching, and composition through logic reasoning techniques. The Semantic Web, based on a description logic-based knowledge representation and reasoning framework, provides the foundations. Integrating aspects from modal logics and process calculi into this framework will prove essential.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Software/Program Verification—*Correctness proofs*; D.2.12 [**Software Engineering**]: Interoperability—*Distributed objects*

General Terms

Design, Verification

Keywords

Web Services, service choreography, ontologies

1. INTRODUCTION

Much progress has very recently been made in the context of the Web Services Framework WSF [31]. The framework itself consists of separate technologies to support the description, discovery and invocation of Web services. Automation and reasoning are central requirements for the success of the WSF in a Web envirnoment.

Various notations and protocols have been suggested to overcome limitations of the core WSF. One aspect building up upon the WSF is the *choreography* of Web services. We use the term *service process* to denote various forms of control flow, conversation, and interaction issues related to the choreography of Web services as they are addressed in flow and conversation languages such as WSFL [16], WSCL [1] or ebXML [31]. An application of service choreography is the development of Web-based business processes.

We extract commonalities of these notations and describe their semantics in a formal, Semantic Web compliant framework [28]. The Semantic Web is a logic-oriented framework addressing limitations of the current Web. Even though being a knowledge representation framework, the Semantic Web and in particular ontologies provide suitable Web technologies for knowledge-related problems arising in Web-based distributed and service-based software development. Semantic Web formalisms guarantee interoperability, acceptance, and support in the Web environment. The need for formality has been recognised for Web services [9, 30], but has not been investigated for Web service choreography.

We look at description, matching, and interaction of Web services that are composed to service processes. Covering the full life cycle of service processes is an essential aspect. Two ontology-based models will form the backbone of a formal process services framework. A *process model* addresses process description and matching, and a *interaction model* addresses service matching and interaction.

Our objective is the integration of formalisms necessary to allow reasoning about service processes into an Semantic Web compliant ontology framework. We start with the presentation of basic aspects of these service processes in terms of classical formalisms such as process calculi [25] and modal logics [15]. The reason for starting off with these two formalisms will become clear when we exploit links to description logic, which underlies Semantic Web ontology languages. For each of these aspects, we will illustrate their representation in ontologies. Here, we will extend our previous work [21, 22] from simple services to service processes. We aim to demonstrate here that an adequate framework exists to carry work on Web service choreography further towards an ontology supported semantic Web service process approach.

We address three central aspects in this paper: the process model, the process life cycle, and the interaction model. After introducing Web service processes and the Semantic Web context in Sections 2 and 3, we focus on the process model by looking at description and matching techniques (Section 4) and ontology support for these (Section 5). Section 6 looks at the life cycle aspects. Section 7 on semantic service matching and Section 8 on ontology support for this

type of matching address the interaction model. We end with related work and some conclusions.

2. SERVICE PROCESSES

The Web Services framework is based on the description, discovery, and use of individual services, each consisting of several operations. However, often more complex tasks are formulated in form of processes. Business processes based on individual, interrelated activities are an example. Some attempts have been made to support the assembly or choreography of services; we summarise these under the term *service processes*. Examples of WSF-based approaches are the Web Service Flow Language WFSL [16] and the Web Service Conversation Language WSCL [1].

Two aspects are central in the assembly of services to service processes in the Web Services context:

- Internal process definition. This addresses the dependencies between the individual activities. Control and data flow can be constrained.
- Process interactions. Based on the Web Services philosophy, each individual activity can be based on an interaction with another service.

The WSFL addresses these two aspects in form of two models. The flow model describes the control flow, i.e. the internal behaviour of processes. The global model describes the interaction behaviour between a service process and other services. Similarly, our investigation will be based on two models, called the *process model* and the *interaction model*.

We will mainly relate our investigation to WSFL. The WSFL motivation is to enable Web Services as implementations for activities of business processes. Services can be composed to service processes. A WSFL process specification consists of an import interface that states the interaction requirements and an export interface that describes the operations offered by the process. This process structure satisfies some major characteristics of a *software component* [27, 6]. The component approach is based on the idea of forming composite reusable software entities based on more basic ones. This component idea can be applied to service processes. However, to support component-style development, some requirements need to be addressed:

- Support of description, discovery, and matching of service processes,
- Support of life cycle aspects including static and dynamic composition.

We address these issues of software service development for the Web and also using the Web in the context of Semantic Web technologies such as ontology frameworks.

3. SEMANTIC WEB AND SOFTWARE DEVELOPMENT

Making the Web more meaningful and open to manipulation by software applications is the Semantic Web objective [4]. Information on the Web shall be made more machine understandable. Annotations shall help software agents to obtain semantic information about documents.

3.1 Semantic Web and Ontology Languages

Expressing meaning through knowledge representation techniques is the starting point of the Semantic Web initiative [28, 4]. A logical framework providing inference rules suitable for automated reasoning is another key ingredient.

For annotations to be meaningful for both creator and user of these annotations, a shared understanding of precisely defined annotations is required. Usually, suitable terminologies and semantical properties are expressed in form of ontologies [12, 14]. Ontologies consist of hierarchical definitions of important concepts in a domain and descriptions of the properties of each concept. Special logics for knowledge representation and reasoning support ontology languages.

The Semantic Web bases the formulation of ontologies on Web technologies for content description – XML and RDF [28]; i.e. the Semantic Web and Web Services share XML technology as the basic layer. Web ontologies can be defined in DAML+OIL – an ontology language rich in description primitives based on XML and RDF/RDF Schema [8].

Formality in the Semantic Web facilitates machine understanding and automated reasoning. DAML+OIL is equivalent to a very expressive *description logic* [13, 3]. This fruitful connection provides well-defined semantics and reasoning systems. Description logics provide a range of class constructors to describe concepts. Axioms express subsumption, equivalence and other properties. Decidability and complexity issues – important for the tractability of the technique – have been studied intensively in the description logic community. Moreover, tools for reasoning exist.

3.2 Ontologies for Software Development

Two ontologies are important for the Web component context:

- *Application domain ontologies* describe the domain of the software application under development.
- *Software ontologies* describe the software development entities such as services.

The need to create a shared understanding for an application domain is long recognised. Client, user and developer of a software system need to agree on concepts for the domain and their properties. Domain modelling is a widely used requirements engineering technique.

With the emergence of distributed software development also the need to create a shared understanding of software entities and development processes arises. We will present basic elements of a software ontology formalising Web services development, in particular providing the crucial matching support for Web service processes.

Some effort has already been made to exploit Semantic Web and ontology technology for the software domain [24, 10, 23]. DAML-S [9] is a DAML+OIL ontology for describing properties and capabilities of Web services. DAML-S represents services as classes (concepts). Knowledge about a service is divided into two parts. A *service profile* is a class that describes what a service requires and what it provides, i.e. external properties. A *service model* is a class that describes properties that concern the service implementation. DAML-S relies on DAML+OIL subsumption reasoning to match requested and provided services.

4. SERVICE PROCESS DESCRIPTION AND MATCHING

Description and matching are design activities. We capture the foundations of this stage in form of a *process model* focussing on the process aspects of service compositions.

4.1 Process Expressions and Interpretation

Languages such as WSFL and WSCL provide means to define execution constraints between services of a process, i.e. define control flow and determine the execution order.

Definition 4.1. **Service process expressions**, *or* **processes** *are inductivley formed based on a set of basic process names, named process expressions, and the combinators sequence ; , parallel composition | , non-deterministic choice + , and iteration ! . A named process expression $P(s_1,\ldots,s_k)$ is defined by a service process expression on based services $s_1,\ldots,s_k$ and the combinators, i.e. expressions such as $P = s_1; s_2; Q$ can be used*[1]*. We often use the notation $P \xrightarrow{s_1;s_2} Q$ for this expression to emphasise the state transition character.*

The process definition is recursive. Based on basic processes (which are Web services), composite services can be defined. We prefer to use the term *process* rather than *service* here.

Example 4.1. *We can specify a business process for an online ordering system*[2]*:*

```
Ordering =
        Login; !(Catalog + Quote); Purchase
```

We can assume an iterated non-deterministic choice of services as the most general, unconstrained case. Without explicit constraints, any of the services can be executed repeatedly. For a process P based on individual services $s_1,\ldots,s_k$ we define a *default process* $!(s_1 + \ \ldots \ + s_k)$.

To support matching of requested and provided processes is our main goal. *Import process patterns* describe how a client process expects to use imported services. *Export process patterns* describe how provided processes have to be used. It can be the case that a client does not need all elements of a business process that are provided.

Processes are composed of individual services. Each of these services is a state transition, i.e. transforms a state of an underlying system into another. The process expressions shall therefore be interpreted in Kripke transition systems KTS [15], a form of labelled transition systems.

Definition 4.2. *Assume a KTS $\{S, L, T, I\}$ consisting of a set of states S, a set of action labels L, a transition relation T on S, and an interpretation I as the* **semantic structure**.

Services P are **interpreted** *as transition relations $P^I \in T$ on $S \times L \times S$. Sequential composition executes one process after the other; the choice operator chooses one process non-deterministically; the iteration iterates the process a non-deterministically choosen (finite) number of times; and parallel composition executes both processes.*

[1]We often drop the list of constituent basic process names.

[2]The example we use throughout this paper is taken from [1].

4.2 Data and Process Interaction

The service process expressions defined so far focus on the control flow, i.e. the execution order. Essential in modelling business processes is to add data flow, i.e. the transfer of data between activations of individual services.

Web Services are connected through a network. The network endpoints that represent services are called *ports* – service names will act as port names. Services (and their ports) can be receivers and senders of data, i.e. read from or write to communication channels set up between the ports.

Definition 4.3. *Assume a service port s and a data item x. Then, $s(x)$ is the* **receive action** *and $\overline{s}\langle x\rangle$ is the* **send action**.

For example, the expression $\overline{\texttt{Quote}}\langle\texttt{prod}\rangle;\texttt{Quote}(\texttt{price})$ asks a service `Quote` for a quote for product `prod` and receives the price `price` in the following action. Data flow internally between services, such as the flow of x in the process $s_1(x);\overline{s_2}\langle x\rangle$, shall not be specified explicity.

Definition 4.4. *An* **interaction** *is the activation of a remote service. Two forms shall be provided:*

- *request-response: for each service s in a service process expression P a write-read sequence $\overline{s}\langle x\rangle; s(y)$ where y is the returned result from an external service.*
- *execute-reply: for each service s in a service process expression P a read-write sequence $s(x); \overline{s}\langle f(x)\rangle$ where f is some internal service functionality.*

These interactions are the basic building blocks of the process life cycle – see Section 6.

All services names in a process expression need to be bound to a concrete service that can execute the service functionality. Finding suitable services that match each individual service requirements and managing the connections is part of the interaction model.

4.3 Matching of Service Processes

The specification of service processes describes the ordering of observable activities of a process. Process calculi such as the π-calculus theory [25] introduce the notion of an experiment to describe observable behaviour of processes. This notion coincides technically with the notion of process descriptions. We will use a notion of simulation between processes to define process matching between requestor and provider. The *requested process* is the input process pattern that the client expects the provider to support.

Definition 4.5. *A provider process P* **matches** *a requested process R if there exists a binary relation $\mathcal{S}$ over the set of processes such that if whenever $R\mathcal{S}P$ and $R \xrightarrow{m} R'$ then there exists P' such that $P \xrightarrow{n} P'$ and $R'\mathcal{S}P'$. We also say that P* **simulates** *R*[3]*.*

The provider needs to be able to simulate the request, i.e. needs to meet the expected request pattern of the client. However, this is not a bisimulation – irrelevant elements in the provider process are not permitted. Dynamic binding of concrete services to the service names is possible. The

[3]The form of this definition originates from the simulation definition of the π-calculus, see e.g. [25].

matching definition is about *potential* interaction. The definition implies that the association between n and m is not fixed. For a given requested service, in principle several different provider services can provide the actual service execution during the process execution.

Example 4.2. *The provider provides a service process*

`Login;!(Catalog+Quote);Purchase`

and the requestor expects support of the process

`!(CatalogBrowse+QuoteProd);ProdPurch`

If the pairs `Catalog/CatalogBrowse`*,* `Quote/QuoteProd`*, and* `Purchase/ProdPurch` *match based on their individual service descriptions – we deal with this type of matching later –, then the provider matches (i.e. simulates) the requested process.*

5. AN ONTOLOGY FOR DESCRIPTION AND MATCHING

5.1 Ontologies for Web Services and Processes

Ontologies are formal frameworks that provide knowledge description and reasoning techniques.

The starting point in defining an ontology is to decide what the basic ontology elements (concepts and roles) represent. Here, the ontology shall formalise a state-transition based software system and its specification.

Definition 5.1. **Concepts** *are classes of objects with the same properties. Concepts are interpreted by sets of objects.* **Individuals** *are named objects. Concepts represent software system properties in this context.*

Systems are dynamic. Descriptions of properties are inherently based on underlying notions of state and state change.

Definition 5.2. **Roles** *in general are relations between concepts. Here, they shall represent two different kinds of relations.* **Transitional roles** *represent services in form of accessibility relations on states, i.e. they represent services resulting in state changes.* **Descriptional roles** *represent properties of a state such as pre- and postconditions or invariant descriptions like service name and description.*

The roles types will turn out to be essential for the matching support for the two different models. Transitional roles are important for process model based matching. Descriptional roles are important for interaction model based matching.

Definition 5.3. **Constructors** *are part of ontology languages that allow more complex concepts to be constructed in form of* **concept descriptions***. Classical constructors include conjunction $\sqcap$ and negation $\neg$. Hybrid constructors are based on a concept and a role – we present these in a description logic notation. The constructor $\forall R.C$ – called* **value restriction** *– is interpreted based on either an accessibility relation R to a new state C for transitional roles, or on a property R satisfying a constraint C for descriptional roles. The dual $\exists R.C$ is called* **existential quantification**.

In Fig. 1, the service process ontology is shown. The state concept shall be introduced as an abstract concept that is described in terms of elements of auxilary domains through descriptional roles such as invariant and mutable state properties (formal conditions, textual descriptions, etc.). The two essential concepts are `pre` and `post`, which denote abstract pre- and post-states for service/process transitions. `preCond` and `postCond` describe states in terms of conditions. For example, $\forall$`postCond.valid(id)` specifies a postState by associating a postcondition `valid(id)`. $\exists$`serv.postCond.true` characterises a preState through a transitional role by stating that there is a service `serv` that can be executed such that the postcondition becomes `true` – which is a termination statement.

We interpret concepts and roles in Kripke transition systems. These are semantic structures used to interpret modal logics that also suffice to interpret description logics.

Definition 5.4. *Assume a Kripke transition system (see Def. 4.2). Concepts are interpreted as sets of states. Transitional roles are interpreted as accessbility relations. Descriptional roles are interpreted as relations involving other concept domains.*

Description logic as the underlying logic of the Semantic Web is particularly interesting for the software engineering context due to a correspondence between description logic and dynamic logic (a modal logic of programs) [26]. This correspondence is based on a similarity between quantified constructors (expressing quantified relations between concepts) and modal constructors (expressing safety and liveness properties of programs). Modal logics such as dynamic and temporal logics have been widely used to specify properties of dynamic and reactive systems.

The formality of the framework allows various logic-based inference and analysis methods to be used. In [22], we introduced a notion of consistency addressing reachability of states for process expressions. We presented criteria for consistency. Deadlock detection techniques can be used to discover mutual dependencies between processes [5].

In the remainder of this section we illustrate how matching of services processes can be represented in a description logic that underlies a Web ontology language.

5.2 A Matching Ontology for Service Processes

We introduced the representation of services in a description logic-based ontology. An ontology that captures service processes and their matching, however, requires an extension of classical description logics [3]. So far, roles – that represent services – are atomic; role constructors to represent service processes have not been provided.

Definition 5.5. *We define the combinators ';' , '!' , '|' and '+' as* **role constructors** *for sequential composition, transitive closure (iteration), intersection (parallel composition without interaction), and union (non-deterministic choice), respectively*[4]*. We also use $\circ$ for sequential composition to emphasise the functional character of roles.*

These role constructors allow us to integrate process description and matching into an ontology framework. A description logic expression such as

$\forall$ `!(Catalog+Browse);Purchase . postState`

is now permitted. We can utilise dynamic logic axioms to reason about process expressions [15].

[4]Usually, the constructors $R \circ S$, R^+, $R \cap S$, and $R \cup S$ are used instead, see [3].

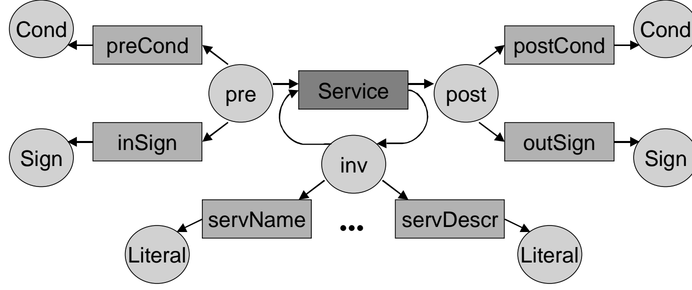

Figure 1: Service Process Ontology

Proposition 5.1. *The following are axioms for the quantified constructors: (i)* $\forall R.\forall S.C \Leftrightarrow R\,;S.C$ *, (ii)* $\forall R.C \sqcap D \Leftrightarrow \forall R.C \sqcap \forall R.D$ *, (iii)* $\forall R \sqcup S.C \Leftrightarrow \forall R.C \sqcup \forall S.C$ *.*

We need to integrate data and process parameters. We introduce data in form of *names*. Names stand for individual data elements. An individual x described by a concept $C(x)$ is interpreted by an element of an underlying state.

Definition 5.6. *We denote a* **name** n *by a role* $n[Name]$*, interpreted by an identify relation* $\{(n^I, n^I)\}$ *for the interpretation* n^I *of* n*. A* **parameterised role** *is a transitional role* R *applied to a name* $n[Name]$*, i.e.* $R \circ n[Name]$[5]*.*

Example 5.1. *For a transitional role* `Login` *and a descriptional role* `postCond`*, the expression*

$$\forall \texttt{Login} \circ \texttt{(id,passwd)}.\forall \texttt{postCond}.\ \texttt{valid(id)}$$

means that by executing `Login∘(id,passwd)` *a post-state can be reached that is described by a postcondition* `valid(id)`*. The term* `Login∘(id,passwd)` *is a composite role expression in which the identifiers* `id` *and* `passwd` *are constant roles (names).*

Earlier on, we distinguished input and output actions, $s(x)$ and $\overline{s}\langle x \rangle$, respectively. These are important for the interactions with actual providers of services. Since matching of processes is here only concerned with control flow patterns, we ignore this distinction here, i.e. the composite role $s \circ x$ abstracts both $s(x)$ and $\overline{s}\langle x \rangle$.

The description logic formula from example 5.1 corresponds to a dynamic (modal) logic formula

$$[\texttt{Login(id,passwd)}][\texttt{postCond()}]\ \texttt{valid(id)}$$

The correspondence allows us to integrate modal axioms and inference rules about processes into description logic.

Subsumption is the central inference technique in description logic.

Definition 5.7. **Subsumption** $C_1 \sqsubseteq C_2$ *of concepts is the subset-relationship* $C_1^I \subseteq C_2^I$ *of the corresponding object classes. Equally, we define subsumption for roles* $R_1 \sqsubseteq R_2$ *as subsets of the corresponding relations* $R_1^I \subseteq R_2^I$*.*

Subsumption reasoning is supported by various axioms.

Proposition 5.2. *For concepts* C_1 *and* C_2 *the following axioms hold: (i)* $C_1 \sqcap C_2 \sqsubseteq C_1$*, (ii)* $C_1 \wedge C_2 \rightarrow C_1$*, (iii)* $C_2 \rightarrow C_1$ *implies* $C_2 \sqsubseteq C_1$*. Analogously for roles.*

[5] We often drop the $[Name]$-postfix when it is clear from the context that a name is referred to.

We will now define service process matching.

Definition 5.8. *A process* $P(n_1, .., n_k)$ **matches** *a process* $R(m_1, .., m_l)$*, if* $P(n_1, .., n_k)$ *simulates* $R(m_1, .., m_l)$*.*

Subsumption on roles is input/output-oriented, whereas the simulation needs to consider internal states of the composite role execution. For each request in a process, there needs to be a corresponding provided service. However, matching is a sufficient condition for subsumption.

Proposition 5.3. *If the process expression* $P(n_1, \ldots, n_k)$ *simulates the process* $R(m_1, \ldots, m_l)$*, then* $R \sqsubseteq P$*.*

6. MATCHING, CONNECTION AND INTERACTION

Description and matching are design activities. Essential is, however, the support of the full process life cycle. Binding individual service names to existing services, i.e. composing a process instance and executing this instance are as important as description and matching.

The foundations of these aspects will be given in form of an *interaction model* that describes bindings, connections, and interactions between services. We extend the matching ontology from Section 5 by life cycle-specific rules.

6.1 Process Life Cycle

Each *service port* s in our process model is actually a family of ports s_C, s_I, s_R that address the needs of the different life cycle stages. Port s_C is a *contract port*, representing an interface that captures abstract properties. s_I and s_R are *connector ports* for interaction – s_I handles the service invocation and input and s_R handles the service reply.

Definition 6.1. *We express the* **service life cycle** *in an annotated process notation – for the* requestor*:*

$$\textsc{Req}\ \overline{s_C}\langle s_I \rangle.!(\ \textsc{Inv}\ \overline{s_I}\langle a, s_R \rangle.\textsc{Res}\ s_R(y).0\ )$$

with annotations for requesting, invoking, and result. Dual to the requestor view there is a provider *view:*

$$\textsc{Pro}\ s_C(s_I).!(\ \textsc{Exe}\ s_I(a, s_R).\textsc{Rep}\ \overline{s_R}\langle f(a) \rangle.0\ )$$

with annotations for providing, executing and replying.

In the requestor view, $\textsc{Req}\ \overline{s_C}\langle s_I \rangle$ is an annotated output action of service s. A process or service can request $\textsc{Req}$ a service using contract port s_C. Connector port references s_I and s_R are subsequently sent for further interactions.

Port types capture properties of a service. If matching between a requestor port type and a provider port type is successful, then a requestor can interact with the provided service repeatedly, expressed using iteration, i.e. it would invoke INV the service at port s_I and receive a result RES at port s_R. The annotations characterise the role of the ports in the life cycle. Each of the individual services of a process goes through the three-port cycle.

Matching and composition activities are captured in a standard life cycle form, which represents a *composition and interaction protocol*. Clients C are parameterised by their required services. Requests have to be satisfied before any interaction can happen. Once a connection is established, a service can be used several times. All service requests need to be satisfied – expressed by a parallel composition | of individual ports:

$$\begin{array}{l} C(m_1,\ldots,m_l) \stackrel{\text{def}}{=} \\ \quad \text{REQ } \overline{m^1_C}\langle m^1_I\rangle.!(\text{INV } \overline{m^1_I}\langle a^1, m^1_R\rangle.\text{RES } m^1_R(y^1).0) \\ \mid \ldots \mid \\ \quad \text{REQ } \overline{m^l_C}\langle m^l_I\rangle.!(\text{INV } \overline{m^l_I}\langle a^l, m^l_R\rangle.\text{RES } m^l_R(y^l).0) \end{array}$$

Service providers P need to be replicated (!) in order to deal with several clients at the same time:

$$\begin{array}{l} P(n_1,\ldots,n_k) \stackrel{\text{def}}{=} \\ !(\ \text{PRO } n^1_C(n^1_I).!(\text{EXE } n^1_I(y^1,n^1_R).\text{REP } \overline{n^1_R}\langle b\rangle.0) \\ \quad + \ \ldots \ + \\ \quad\ \text{PRO } n^k_C(n^k_I).!(\text{EXE } n^k_I(y^k,n^k_R).\text{REP } \overline{n^k_R}\langle b\rangle.0)\ ) \end{array}$$

Providers do not need to engage in interactions with all their ports – modelled by using the choice operator + instead of the parallel composition | of client services.

Clients and servers are composed in parallel to form a system. A process is often both client and provider of services. Requirements REQ $m^i_C\langle m^i_I\rangle$ $(i = 1,..,l)$ have to be satisfied and connectors have to be established, before any service PRO $n^j_C(n^j_I)$ $(j = 1,..,k)$ can be provided.

6.2 Service Connection and Interaction

Composition consists of two activities: matching and connection. Successful matching can result in a connection between services. So far, we have been looking at matching of abstract process descriptions, resulting in a support technique formalised as a matching ontology. We now focus on the computational side of compositions. The connection of matching services shall now be formalised using an operational semantics.

In the composition process we can distinguish a *contract phase* where both process instances try to form a contract based on abstract descriptions. The *connection phase* establishes a connector channel for interaction between the services. We will capture contract and connector establishment in form of transition rules – Fig. 2.

For a composition $\overline{m_C}\langle m_I\rangle.C|n_C(n_I).P$ of a client and a provider, both processes commit themselves to a communication along the channel between ports m_C and n_C, if their specifications match. A *contract rule* formalises the process of matching and commitment, see Fig. 2[6]. We will explain the details of the *type*-notion in the next section. The arrows $\rightarrow$ denote state transitions of the individual processes, either through observable actions $\overline{x}\langle y\rangle$ and $x(y)$ or through silent, non-observable *interactions* τ. We define a *composition* $C\frown P$ by $\nu c(\{c/m_I\}C|\{c/n_I\}P)$[7] where c represents a private channel – the connector, that replaces the port names for the interaction. The composition yields a proper process – based on two service processes composed in parallel. The synchronisation and interaction between the processes is guarded by type-based matching constraints.

Example 6.1. *The user requires a service (annotation* REQ*) through port* $\texttt{Quote}_C$ *and the server provides a service (annotation* PRO*) through port* $\texttt{QuoteProd}_C$*:*

$$\begin{array}{l} \texttt{Req} \stackrel{def}{=} \text{REQ } \overline{\texttt{Quote}_C}\langle\texttt{Quote}_I\rangle.\texttt{Req}' \\ \texttt{Pro} \stackrel{def}{=} \text{PRO } \texttt{QuoteProd}_C(\texttt{QuoteProd}_I).\texttt{Pro}' \end{array}$$

A connector is created if a client requesting m_I invokes a service n_I at the server side, described by the *connector rule*, see Fig. 2. Parameter data a and a reply channel m_R are sent to the provider. The types $type_C(m_I)$ and $type_C(n_I)$ represent the connector activation types.

Example 6.2. *The composition of* Pro' *and* Req' *creates a connector that allows the client to use a service, e.g.* QuoteProd*, provided by the server.*

$$\begin{array}{l} \texttt{Req}' \stackrel{def}{=} \text{INV } \overline{\texttt{Quote}_I}\langle\texttt{pid}\rangle.\texttt{Req}'' \\ \texttt{Pro}' \stackrel{def}{=} \text{EXE } \texttt{QuoteProd}_I\texttt{(x)}.\texttt{Pro}'' \end{array}$$

*The requestor can invoke (*INV*) a service through the interaction port* $\texttt{Quote}_I$*, which will trigger the execution (*EXE*) of* $\texttt{QuoteProd}_I$ *with parameter* pid *by the server.*

6.3 Ontology Support for Interaction

We have formulated the operational semantics of interaction in form of process calculus-style reaction rules. In terms of the ontology, services were so far described as transitional roles and we considered system states that describe service (and process) properties such as pre- and postStates to define input/output behaviour.

We will now formalise composition and interaction in the ontology through inference rules. In order to address interaction in the ontological framework, we need to look at a special kind of a parallel composition transition. This transition is based on the *synchronisation* of concurrent services through data exchange. Usually, a special form of transition – here denoted by τ – is used to denote this synchronisation.

We can characterise properties of these interactions between two services, here a reformulation of the contract rule without annotations and matching constraints,

$$\frac{\overline{m_C}\langle m_I\rangle.p_{m_C} \stackrel{\overline{m_C}\langle m_I\rangle}{\longrightarrow} p_{m_C} \qquad n_C(n_I).p_{n_C} \stackrel{n_C(n_I)}{\longrightarrow} p_{n_C}}{\overline{m_C}\langle m_I\rangle.p_{n_C} + M_1 | n_C(n_I).p_{n_C} + M_2 \ \stackrel{\tau}{\longrightarrow}\ p_{m_C} \frown p_{n_C}}$$

in terms of the ontology language by an inference rule:

$$\frac{\forall\ m_C \circ m_I\ .\ post_{m_C} \qquad \forall n_C \circ n_I\ .\ post_{n_C}}{\forall\ m_C \circ m_I | n_C \circ n_I\ .\ post_{m_C} \sqcap post_{n_C}}$$

This inference rule complements other process constructor specific axioms and rules that we can derive from dynamic logic and process calculi, see Prop. 5.1 and Def. 5.8 These

[6]The contract rule differs from the π-calculus reaction rule which requires channel names to be the same [25]. We only require a subtype relationship between ports. Type systems for the π-calculus usually constrain data that is sent; we constrain reaction, i.e. the interaction between agents.

[7]The substitution $\{b/a\}$P means that b replaces a in P.

$$\text{(a)} \quad \frac{\textsc{Req}\ \overline{m_C}\langle m_I\rangle.C \xrightarrow{\overline{m_C}\langle m_I\rangle} C \quad \textsc{Pro}\ n_C(n_I).P \xrightarrow{n_C(n_I)} P}{\textsc{Req}\ \overline{m_C}\langle m_I\rangle.C+M_1|\textsc{Pro}\ n_C(n_I).P+M_2 \xrightarrow{\tau} C\frown P}\langle\ type(n_C)\leq type(m_C)$$

$$\text{(b)} \quad \frac{\textsc{Inv}\ \overline{m_I}\langle a,m_R\rangle.C \xrightarrow{\overline{m_I}\langle a,m_R\rangle} C \quad \textsc{Exe}\ n_I(x,n_R).P \xrightarrow{n_I(x,n_R)} P}{\textsc{Inv}\ \overline{m_I}\langle a,m_R\rangle.C+M_1|\textsc{Exe}\ n_I(x,n_R).P+M_2 \xrightarrow{\tau} C\frown\{a/x\}P}\langle\ type(n_I)\leq type(m_I)$$

Figure 2: Contract Rule (a) and Connector Rule (b)

axioms and inference rules form an application-specific extension of description logic that allow us to infer more properties about service processes and their interactions.

7. SEMANTIC SERVICE MATCHING

The remaining issue is matching of individual services for service-level interactions, which we will capture in the *interaction model*. The description of services should include behavioural aspects (e.g. pre- and postconditions), but also non-functional descriptions such as the author or a description – see Fig. 1. We focus on abstract behaviour here.

DAML-S [9] is an example of an ontological framework that supports matching of semantically described services. Our approach, however, is different from DAML-S. We discuss this difference in more detail in Section 8. We present an integrated and coherent framework that includes service process description and matching. Based on the correspondence with dynamic logic, our reasoning about services and processes is improved.

7.1 Semantic Service Description

The basic WSF building blocks are *ports*, which represent services. WSF port types define services based on input and output messages. Services, however, should be described by various properties – functional and non-functional – such as author, signatures, and abstract behaviour. We extend the WSF port type specification by contractual information capturing service semantics.

Definition 7.1. **Port types** *describe the expected capacity (the kind of entities that can be sent):* $type(s_C)$ *is a contract type for a contract port including a pre- and a postcondition*[8]*,* $type(s_I)$ *of connector port* s_I *is a function type consisting of parameter types for data and reply channel.*

We use pre- and postconditions as abstractions for ports [29], enabling the design-by-contract approach [18]. Dynamic logic is a suitable logical framework that subsumes pre- and postcondition specification [15]. This is one of the reasons why the connection between description logic and modal logic is so important in this context.

Example 7.1. *A requirements specification of a service user for a* `Login` *service:*

service `Login(id:ID,passwd:Pass)`
  pre `syntaxOK(id)`
  post `valid(id)` ∨ `invalid(id)` ∨ `unknown()`

[8]More aspects, e.g. signatures or invariants, would need to be considered in a complete and comprehensive treatment. Signatures, for example, could determine the structural compatibility of port types.

A service provider specification for a `UserLogin` *service:*

service `UserLogin(id:ID,passwd:Pass)`
  pre `true`
  post `valid(id)` ∨ `invalid(id)`

We have used a simple contract idea here to illustrate the technique; in practice a more advanced variant might be used [32]. Preconditions constitute provision declarations rather than requirements for the client. Consequently, clients often do not specify them in their strongest form.

7.2 Matching of Interacting Services

We can express service contracts in dynamic logic, e.g. `syntaxOK(id)` → [`Login(id,passwd)`] `valid(id)`, using the modal box operator for this safety condition. If the precondition `syntaxOK(id)` holds, then `Login` can be executed such that a postState satisfying `valid(id)` can be reached.

Definition 7.2. *Two services described by pre- and postconditions and represented by contract ports* n_C *and* m_C **match***, expressed* $type_c(n_C) \leq type_c(m_C)$*, if the precondition is weakened and the postcondition strengthened.*

This definition is derived from an inference rule of the dynamic logic – the consequence rule CONS – which expresses refinement of programs [15, 32].

Example 7.2. *The provided service* `UserLogin` *matches the requirements of* `Login` *in Example 7.1.* `UserLogin` *has a weaker, less restricted precondition (*`syntaxOK(id)` *implies* `true`*) and a stronger postcondition (the disjunction* `valid(id)`∨`invalid(id)` *implies* `valid(id)`∨`invalid(id)`∨ `unknown()`*. This means that the provided service satisfies the requirements; it is even better than requested.*

Another reason to choose dynamic logic as the framework for functional properties is to exploit the logic's expressive power to specify both safety and liveness properties of processes and their interactions. Dynamic logic is a rich framework – we have used only one simple rule here.

Example 7.3. *We can express that after executing the login service, a product will eventually be purchased,*

[`Login(id,passwd)`]⟨`Purchase(prod)`⟩ `true`

combining safety ($[\ldots]\phi$*) and liveness (*$\langle\ldots\rangle\psi$*) properties.*

7.3 Ontologies

A service is functionally specified through pre- and postconditions. Matching of services has been defined in terms of implications on pre- and postconditions, and has been represented as a subtype relation between the contract ports.

Subsumption $\sqsubseteq$ is the central reasoning concept in description logics, i.e. we need to integrate reasoning about services matching into this approach. Subsumption is here interpreted as a subset relationship on sets of states that satisfy pre- or postState descriptions. We present an inference rule for subsumption of transitional roles that can be proved correct through a subset relationships on sets of states.

Definition 7.3. *We define the* **matching** *inference rule*

$$\frac{\forall preCond.pre_P \sqcap \forall P.\forall postState.post_P}{\forall preCond.pre_R \sqcap \forall R.\forall postState.post_R} \left\langle \begin{array}{l} pre_P \sqsubseteq pre_R, \\ post_R \sqsubseteq post_P \end{array} \right.$$

for transitional roles[9] *P and R.*

The modal CONS-rule, used in dynamic logic to express refinement of programs, serves here to derive this rule.

Proposition 7.1. *The matching rule for services defined in Definition 7.3 is sound.*

Matching implies subsumption, but is not the same. Matching of services is a sufficient criterion for subsumption.

Proposition 7.2. *If service P matches R, then* $P \sqsubseteq R$.

If pre- and postconditions represent application domain-specific formulas, e.g. `valid(doc)`, then an underlying domain-specific theory, which could be provided by an application domain ontology, is needed. Even though description logics can form the core of a matching ontology, in our context description logic reasoning might need to be combined with reasoning in other formal theories.

8. RELATED WORK

Some frameworks for advanced services architectures on the Web have already been proposed. Fensel and Bussler [11] present a framework for Web-based services, called *Web Services Modelling Framework* (WSMF). The development of the framework focussing on the integration of semantic Web technology is in progress. The issue of composed Web services is addressed in WSFL [16]. Business processes and interactions are the two types of processes that result in the composition of services. We took this language as the starting point to integrate formal foundations into the Web Services context. DAML-S [9] provides to some extend for Web services what we aim at for Web service processes. However, the form of reasoning and ontology support that we provide here is not possible in DAML-S, since services are modelled as concepts and not rules in the DAML-S ontology. Only considering services as roles makes modal reasoning about process behaviour possible in a description logic framework. [19] investigates the composition of Web services within the DAML-S context. Composite processes descriptions are integrated into PSL – the Process Specification Language – an ontology for process specifications formalised in the situation calculus, which is used to represent dynamically changing worlds.

Software architecture addresses problems that arise when systems are constructed from components. Components are identified as primary points of computation and connectors define interactions between the compoments. Darwin [17] is a structural configuration language that allows to specify the architecture of distributed and interacting component-based systems. The language Wright [2] addresses similar problems. This coincides with our aim to support process composition and interaction. A basic component model underlies the WSF [7]. The authors state that strengthening the component aspect in WSF will improve the framework – our composable service processes are a first step in this direction.

A formally defined model for Web service processes is essential if analysis and reasoning services based on semantic descriptions shall be provided. In [21, 22], we have presented a formal framework for component composition based on a typed process calculus targeting a component model similar to that of basic services. Typed process models to formalise interaction have also been used elsewhere. Nierstrasz [20] develops a formal type-theoretic framework for objects. Objects are characerised as regular processes that interact with each other. Similar to our two models, a two-layered type system distinguishes services types (contracts) and regular types (protocols). Two subtype notions – based on services types and regular types – define a notion of satisfiability between client and provider. The orthogonality of the two forms of types is emphasised.

9. CONCLUSIONS

A developer of services and service processes needs composition techniques, i.e. needs support to discover, match, and integrate existing services and processes into a service process under development. We have presented composition techniques for semantic Web service processes that are interoperable with current Semantic Web technology. Our matching and connection techniques are based on rigorous foundations that provide precise semantics and allow reasoning about properties of service processes. We can summarise our results as a two-layered approach:

- An upper, abstract layer supports matching techniques. Our central achievement is the integration of description and matching techniques – addressing process properties and interaction patterns – into a description logic-based ontology, utilising a correspondence between description logics and dynamic logic to encode program logic principles in description logic.
- A lower layer describes full details of the life cycle stages and interactions of process instances. It consists of observable behaviour of services, abstracted by dynamic logic , which is incorporated into the ontological framework through inference rules. This is an essential foundation for structural configuration tasks in a Web middleware architecture.

We have developed an ontological framework for Web service processes based on classical formalisms such as process calculi and modal logics. We have exploited links and similarities between them and description logic, which allowed us to integrate results from these rich areas into the service process ontology.

The example we used here is taken from [1]. We feel that it illustrates the support that our framework can provide for describing, matching, and composing service processes.

An important characteristics is the adherence to Web standards, which provides interoperability with Web techniques and tools, and increases the acceptance. The combination

[9] Descriptional roles are treated in the usual style.

of a software ontology with application domain ontologies makes this even necessary. Another important aspect is the need for a high degree of formality. The Semantic Web will incorporate logic and reasoning, aiming at automation and unambiguous shared understanding.

Some questions have remained unanswered. We have used a very expressive description logic – which requires more investigations into decidability and complexity. Besides these fundamental questions, practical aspects such as broker architectures and tools need to be addressed.

10. REFERENCES

[1] A. Banerji et.al. *Web Services Conversation Language.* http://www.w3.org/TR/wscl10/, 2003.

[2] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Trans. on Software Engineering and Methodology*, 6(3):213–249, 1997.

[3] F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5), May 2001.

[5] A. Brogi, E. Pimentel, and A.M. Roldán. Compatibility of Linda-based Component Interfaces. In A. Brogi and E. Pimentel, editors, *Proc. ICALP Workshop on Formal Methods and Component Interaction.* Elsevier Electronic Notes in Theoretical Computer Science, 2002.

[6] I. Crnkovic and M. Larsson, editors. *Building Reliable Component-based Software Systems.* Artech House Publishers, 2002.

[7] F. Curbera, N. Mukhi, and S. Weerawarana. On the Emergence of a Web Services Component Model. In *Proc. 6th Workshop on Component-Oriented Programming WCOP'01.* 2001.

[8] DAML Initiative. *DAML+OIL Ontology Markup.* http://www.daml.org, 2001.

[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] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Semantic Configuration Web Services in the CAWICOMS Project. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.

[11] D. Fensel and C. Bussler. The Web Services Modeling Framework. Technical report, Vrije Universiteit Amsterdam, 2002.

[12] M. Gruninger and J. Lee. Ontology – Applications and Design. *Communications of the ACM*, 45(2):39–41, Feb 2002.

[13] I. Horrocks, D. McGuiness, and C. Welty. Digital Libraries and Web-based Information Systems. In F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors, *The Description Logic Handbook.* Cambridge University Press, 2003.

[14] H. Kim. Predicting How Ontologies for the Semantic Web Will Evolve. *Communications of the ACM*, 45(2):48–54, Feb 2002.

[15] Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier Science Publishers, 1990.

[16] F. Leymann. Web Services Flow Language (WSFL 1.0), 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[17] 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)*, volume 989, pages 137–153. Springer-Verlag, 1995.

[18] Bertrand Meyer. Applying Design by Contract. *Computer*, pages 40–51, October 1992.

[19] S. Narayanan and S.A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. World-Wide Web Conference WWW'2002.* ACM, 2002.

[20] Oscar Nierstrasz. Regular types for active objects. In *Proceedings OOPSLA '93, ACM SIGPLAN Notices*, pages 1–15, October 1993.

[21] C. Pahl. A Formal Composition and Interaction Model for a Web Component Platform. In A. Brogi and E. Pimentel, editors, *Proc. ICALP Workshop on Formal Methods and Component Interaction.* Elsevier Electronic Notes in Theor. Computer Science, 2002.

[22] C. Pahl. An Ontology for Software Component Matching. In *Proc. Fundamental Approaches to Software Engineering FASE'2003.* Springer-Verlag, LNCS Series, 2003.

[23] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In I. Horrocks and J. Hendler, editors, *Proc. 1st Int. Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer, 2002.

[24] J. Peer. Bringing Together Semantic Web and Web Services. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer, 2002.

[25] D. Sangiorgi and D. Walker. *The π-calculus - A Theory of Mobile Processes.* Cambridge University Press, 2001.

[26] K. Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. 12th Joint Conference on Artificial Intelligence.* 1991.

[27] C. Szyperski. *Component Software: Beyond Object-Oriented Programming – 2nd Ed.* Addison-Wesley, 2002.

[28] W3C Semantic Web Activity. Semantic Web Activity Statement, 2002. http://www.w3.org/sw.

[29] J.B. Warmer and A.G. Kleppe. *The Object Constraint Language – Precise Modeling With UML.* Addison-Wesley, 1998.

[30] World-Wide Web Conference WWW'2003. *Semantic Web Services Panel.* ACM, 2003.

[31] World Wide Web Consortium. *Web Services Framework.* http://www.w3.org/2002/ws, 2003.

[32] A. Moorman Zaremski and J.M. Wing. Specification Matching of Software Components. *ACM Trans. on Software Eng. and Meth.* 6(4):333–369. 1997.