An Ontology-based Framework for Semantic Grid Service Composition

Claus Pahl

Dublin City University, School of Computing
Dublin 9, Ireland
Claus.Pahl@dcu.ie

Abstract. The Semantic Grid aims at enhancing Grid architectures by knowledge engineering techniques. The service notion is central in this approach. Service-level agreements, called contracts, are formed to define the service usage conditions. Ontology technology can form the framework to capture semantical conditions for contracts. Often, applications of Grid services involve a combination of several services. We present an ontology-based framework for service composition for the Semantic Grid. We take a process-oriented view of services, achieving an intrinsic representation of services and their composition in an ontology.

Keywords: Semantic Grid, Service composition, Ontology, Service processes.

1 Introduction

Knowledge is expected to become more central in Grid architectures [1, 2]. The Semantic Grid aims at enhancing Grid architectures by knowledge engineering techniques. Semantic Web ontologies can support this endeavour [3, 4].

The service notion is central in this approach. We view Grid architectures as sets of services [1, 5–7]. Services are provided to Grid users. Service-level agreements, called contracts, define the service usage conditions. Ontology technology can form a marketplace framework to capture semantical conditions for contracts in a common, shared format. Grid service applications are often complex. Services need to be composed to achieve a complex goal. Two aspects characterise our approach. Firstly, services shall be considered as processes – services can be viewed from an external perspective as interacting agents in a distributed Grid environment [8]. Secondly, a composition language based on this process view can enable service interoperability for the Grid.

Our aim here is to develop a Semantic Web-based Grid service ontology that acts as service composition framework. An ontology framework can enable knowledge representation for the Grid, for instance for the representation of service contract agreements between provider and user of services. We will introduce a composition language integrated with a knowledge representation framework.

Reasoning about service descriptions and service matching to identify suitable services in marketplace and to define consistent service compositions is an

important activity. We will present here a services development ontology that provides matching support [9]. Ontologies are knowledge representation frameworks defining concepts of a domain and their properties; they provide the vocabulary and facilities to reason about these. Ontologies provide a shared and agreed knowledge infrastructure. Two types of ontologies are important for the Grid services context. Application domain ontologies describe the domain of the grid software under development. Software ontologies describe the service entities and aspects of the development and deployment life cycle.

Formality in the Semantic Web framework facilitates machine understanding and automated reasoning – automation is essential for the future Grid. The ontology language OWL is equivalent to a very expressive description logic [10], which provides well-defined semantics and reasoning systems. It has already been applied to Grids [2].

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. However, with the emergence of distributed software development such as Grids and service-oriented architectures also the need to create a shared understanding of software entities and development processes arises.

We introduce the background including Grid architectures, services, and ontologies in Section 2. In Section 3, we define a simple ontology language for Semantic Grid service description. The description of composed services is subsect of Section 4. In Section 5, we address matching of service processes. We discuss the wider context of semantic services for the Grid and related work in Section 6. We end with some conclusions.

2 Semantic Grid Services and Architectures

2.1 Grid Architectures

Grid technology aims at supporting sharing and coordinated use of resources in dynamic, distributed virtual organisations [5]. In the *Open Grid Service Architecture* [5], a Grid service is defined as a Web service that provides a set of well-defined interfaces and that follows specific conventions. Aspects that are Grid-specific in the Web services context are:

- *statefulness* – services often encapsulate a hidden state,
- *dynamic assembly* and *transient character*,
- *upgradeability* – due to the dynamic nature, change management is essential.

Grids can be described as *layered architectures* with three conceptual layers [1]:

- *Data and computation*: This layer deals with the allocation, scheduling, and execution of computational resources.
- *Information*: This layer deals with representation, storage, access, sharing, and maintenance of information, i.e. data associated with its semantics.

– *Knowledge*: This layer deals with the acquisition, usage, retrieval, publication, and maintenance of knowledge, i.e. information capturing goals, problems, and decisions.

We will demonstrate that ontologies not only support the information and knowledge layer, but that ontologies can also help to integrate the computational aspects of the lower data and computation layer.

It is important to note that our ontological framework is not specific to any of the layers or specific services. We will introduce an abstract framework suitable for the composition of any services – no matter on which layer.

2.2 A Basic Services Model and Service-related Activities

The composition of Grid services to address complex goals and to form higher-level applications requires a service model that captures the essential characteristics of services for this context. Descriptions, contracts, and compositions determine the requirements for such a service model [11]:

– *Explicit export and import interfaces.* In particular explicit and formal import interfaces make services more context independent. Only the properties of required and provided services are specified.
– *Contractual (semantic) description of services.* In addition to syntactical information such as service signatures, the abstract specification of behaviour (a contract) is a necessity for reusable services.
– *Service protocol.* An interaction protocol describes the ordering of service activations that a user of a service has to follow in order to use the service in a meaningful and consistent way.

Our aim is to capture composition in form of *processes* – expressions that describe *business processes* and *workflows* through ordering dependencies and invocations of services. The consistency of a process would depend on individual service descriptions based on the service model.

Three *development activities* are essential in this context – which we will address in the subsequent three sections:

– *Description.* An ontology language will allow us to describe individual services in a Semantic Web-compatible framework.
– *Composition.* An extension of the ontology will allow composition of services to higher-level services to be expressed.
– *Matching.* Inference capabilities of the ontological framework will be used to reason about matching between provided and required service processes.

The activities are essential parts of the *lifecyle* of a Grid service [1].

2.3 Ontology Technology

Ontologies are means of knowledge representation, defining so-called shared conceptualisations. Ontologies are frameworks for terminological definitions that can

Service `DocumentStorageServer`
provided services
`crtDoc(id:ID)`
`rtrDoc(id:ID):Doc`
`updDoc(id:ID,upd:Doc)`
`delDoc(id:ID)`
contract information
`updDoc(id:ID,upd:Doc)`
preCond `wellFormed(upd)`
postCond `rtrDoc(id)=upd` $\wedge$ `wellFormed(upd)`
service interaction protocol
`crtDoc;!(rtrDoc+updDoc);delDoc`

Fig. 1. A Sample Application: a Document Storage Service.

be used to organise and classify concepts in a domain. Combined with a symbolic logic, we obtain a framework for specification, classification, and reasoning in an application domain. Terminological logics such as description logics [10] are an example of symbolic logics.

The Semantic Web is an initiative for the Web that builts up on ontology technology [3]. XML is the syntactical format. RDF – the Resource Description Framework – is a triple-based formalism (subject, property, object) to describe entities. OWL – the Web Ontology Language – provides additional logic-based reasoning based on RDF.

We will use Semantic Web-based ontology concepts to formalise and axiomatise Grid service processes, i.e. to make statements about services and to reason about them. We will base our ontology on Description logic [10]. Description logic, which is used to define OWL, is based on concept and role descriptions. *Concepts* represent classes of objects; *roles* represent relationships between concepts; and *individuals* are named objects. Concept descriptions are based on primitive logical combinators (negation, conjunction) and hyprid combinators (universal and existential quantification). Expressions of a description logic are interpreted through sets (concepts) and relations (roles).

Description logic is particularly interesting for the software development context due to a correspondence between description logics and modal logic [12, 13]. This will allows us to embed modal reasoning about processes in a description logic context – achieving an intrinsic specification of processes.

2.4 An Example

An example shall illustrate our service ontology – see Fig. 1. It describes a *document storage and access service*. The service is an example for a *data and computational layer Grid service*. The `DocumentStorageServer` service allows users to create, retrieve, update, and delete documents.

– An empty document can be created using `crtDoc`. The service `rtrDoc` retrieves a document, but does not change the state of the server component, whereas the update service `updDoc` updates a stored document without returning a value. Documents can also be deleted.
– We have illustrated contract-related information by specifying one of the operations by pre- and postcondition. If documents are XML-documents, these can be well-formed (correct tag nesting) or valid (well-formed and conform to a schema definition).
– The interaction protocol defines an ordering constraint that has to be obeyed if the service is to be used.

A service user might need the following services to assemble a higher-level service:

`create(id:ID)`, `retrieve(id:ID):Doc`, and `update(id:ID,upd:Doc)`

The user might require `create;!(retrieve+update)` to implement a goal or business process. The `create` service is expected to be executed first, followed by a repeated invocation of either `retrieve` or `update`.

3 Description of Semantic Grid Services

Dynamic assembly and management of Grid services rely on a high degree of automation. Semantical information about services can support automation. Semantics equally support upgradeability. Backward compatibility is required for Grid Service architectures. A semantical framework can capture explicit constraints to maintain integrity during change management.

3.1 An Ontology for Service Description

The starting point in defining an ontology is to decide what the basic ontology elements – concepts and roles – represent. An intuitive idea would be to represent services as concepts. Our key idea, however, is that the ontology formalises a software system and its specification, see Fig. 2.

– Concepts – circles in the diagram – shall represent static Grid system descriptions such as invariants and/or other syntactical and semantical aspects. Importantly, systems are dynamic, i.e. the descriptions of properties are inherently based on an underlying notion of state and state change.
– Roles – rectangles in the diagram – shall represent two different kinds of relations. *Transitional roles* represent accessibility relations, i.e. they represent processes resulting in state changes. *Descriptional roles* represent properties in a given state, i.e. static relations.

A language based on these constructs introduces a general terminological framework. Central here is the notion of states that capture properties of a system and a service. We will focus on functional properties here; non-functional aspects could be integrated as invariant (*inv*, see Fig. 2) properties[1].

[1] Ontological frameworks for semantic Web services such as OWL-S [9] provide this type of support.

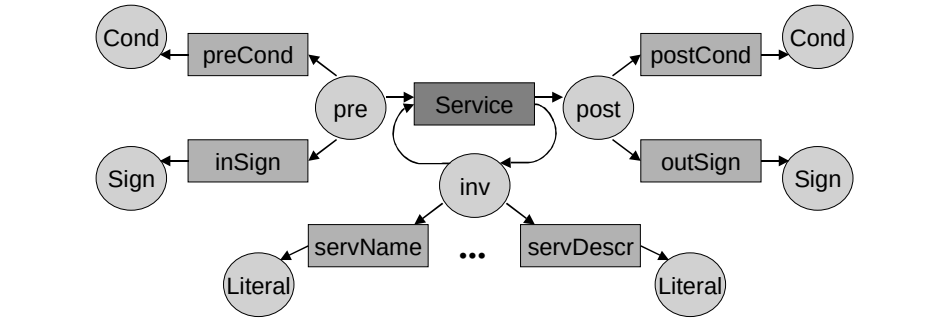

Fig. 2. Semantic Grid Services Ontology

3.2 A Basic Ontology Language – Syntax

We develop a description logic to define our service description, composition, and matching ontology. A description logic consists of three types of entities. Individuals can be thought of as constants, concepts as unary predicates, and roles as binary predicates.

Concepts are the central entities. They can represent anything from concrete objects of the real world to abstract ideas. Constructors are part of ontology languages that allow more complex concepts (and roles) to be constructed. Classical constructors include conjunction and negation. Hybrid constructors are based on a concept and a role – we present these in a description logic notation.

- **Concepts** are collections or classes of objects with the same properties. Concepts are interpreted by sets of objects.
- **Roles** are relations between concepts.
- **Individuals** are named objects.
- **Concept descriptions** are formed according to the following rules: A is an atomic concept, and if C and D are concepts, then so are $\neg C$ and $C \sqcap D$. Combinators such as $C \sqcup D$ or $C \rightarrow D$ are defined as usual.

Roles allow us to describe a concept through its relationship to other concepts. Two basic forms of role applications are important for our context. These will be made available in form of concept descriptions. Value restriction and existential quantification extend the set of concept descriptions.

- A **value restriction** $\forall R.C$ restricts the value of role R to elements that satisfy concept C.
- An **existential quantification** $\exists R.C$ requires the existence of a role value.

Quantified roles can be composed. Since $\forall R_2.C$ is a concept description, the expression $\forall R_1.\forall R_2.C$ is also a concept description.

The constructor $\forall R.C$ is interpreted as either an accessibility relation R to a new state C for transitional roles such as `update`, or as a property R satisfying a constraint C for descriptional roles such as `postCond`.

3.3 A Basic Ontology Language – Interpretation

We interpret concepts and roles in Kripke transition systems [13]. Kripke transition systems are semantical structures used to interpret modal logics that are also suitable to interpret description logics [10]. A **Kripke transition system** (KTS) $M = (\mathcal{S}, \mathcal{L}, \mathcal{T}, I)$ consists of a set of states $\mathcal{S}$, a set of role labels $\mathcal{L}$, a transition relation $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$, and an interpretation I.

We use Kripke transition systems to facilitate the transitional character of service-based Grid systems. Concepts are interpreted as states. Transitional roles are interpreted as accessibility relations. The set $\mathcal{S}$ interprets the state domains *pre*, *post*, and *inv* – see Fig. 2. We can extend the set $\mathcal{S}$ of states by several auxiliary domains such as *Cond*, *Sign* or *Literal* or other aspects that capture contract-specific properties. *Cond* represents conditions or formulas, *Sign* denotes a domain of service signatures and *Literal* denotes string literals.

For a given Kripke transition system M with interpretation I, we define the model-based **semantics of concept descriptions**[2]:

$$\begin{aligned}(\neg A)^I &= \mathcal{S} \backslash A^I \\ (C \sqcap D)^I &= C^I \cap D^I \\ (\forall R.C)^I &= \{a \in S | \forall b.(a,b) \in R^I \rightarrow b \in C^I\} \\ (\exists R.C)^I &= \{a \in S | \exists b.(a,b) \in R^I \wedge b \in C^I\}\end{aligned}$$

Expressive role constructs are essential for our application. We distinguish

- **transitional roles** $R_{\mathcal{T}}$ that represent component services: $(R_{\mathcal{T}})^I \subseteq \mathcal{S} \times \mathcal{S}$. They are interpreted as accessibility relations on states.
- **descriptional roles** $R_{\mathcal{D}}$ that are used to describe properties of services dependant on the state: $(R_{\mathcal{D}})^I \subseteq \mathcal{S} \times \mathcal{D}$ for some auxiliary domain $\mathcal{D}$. These are interpreted as relations between states and property domains.

Some predefined roles, e.g. the identity role *id* interpreted as $\{(x,x) | x \in \mathcal{S}\}$, shall be introduced. The predefined descriptional roles are defined as follows:

$$\begin{aligned} preCond^I &\subseteq pre^I \times Cond^I & postCond^I &\subseteq post^I \times Cond^I \\ inSign^I &\subseteq pre^I \times Sign^I & outSign^I &\subseteq post^I \times Sign^I \\ servName^I &\subseteq inv^I \times Literal^I & servDescr^I &\subseteq inv^I \times Literal^I \end{aligned}$$

Note, that these descriptional roles are part of the wider ontology framework – for the remainder of the paper, we will concentrate on transitional roles.

4 Composition of Grid Services

Composition of services to *higher-level services* becomes a central activity in distributed computational environments, if reuse and sharing is an objective [14]. We introduce a notion of *service processes* for two reasons:

[2] The semantics of description logics is usually given by interpretation in models. However, it can also be defined by translation into first-order logic [10]. Concepts C can be thought of as unary predicates $C(x)$. Roles R can be thought of as binary relations $R(x,y)$. Then, $\forall R.C$ corresponds to $\forall x.R(y,x) \rightarrow C(x)$.

– *Provider side*: Services are sets of operations. An ordering of these operations, specified through an *interaction protocol*, is often necessary to guarantee a coherent usage of the service. Grid services are often stateful. Lifecycle constraints do often apply, e.g. a *create*-operation might need to be invoked before any other functionality can be used.
– *Client side*: Service usage is not restricted to request-response interactions. Service usage can be based on a composition of individual services or service operations to complex *higher-level processes* to satisfy user needs. Grid service architectures are often based on a factory service that might create multiple service instances and combine these to a higher-level service. A process-style description can define higher-level services.

Providing a framework that represents these service processes is required.

4.1 Service Process Expressions

An ontology supporting service composition requires an extension of basic description logics by *composite roles* that can represent *service processes* [10]. These are necessary to express *interaction protocols* for a single service and to define composed *higher-level services*.

The following **role constructors** shall be introduced for **service process composition**:

– $Q\,;R$ **sequential composition** with $(Q\,;R)^I = \{(a,c) \in S^I \times S^I | \exists b.(a,b) \in Q^I \wedge (b,c) \in R^I\}$; often we use $\circ$ instead of ; for functional composition
– $!R$ **iteration** with $!R^I = \bigcup_{i \geq 1} (R^I)^i$, i.e. the transitive closure of R^I
– $Q + R$ **non-deterministic choice** with $(Q+R)^I = Q^I \cup R^I$

Each service process is assumed to be sequential; concurrently executed services are, however, possible. This language, which defines role expressions, is a regular language. This property might be useful if finite state machine or automata-based approaches for analysis are used. Two additional constructs:

– Often, we want to express the sequential composition of functional roles. A **role chain** $R_1 \circ \ldots \circ R_n$ is a sequential composition of functional roles (roles that are interpreted by functions).
– $A(R_1, \ldots, R_n)$ is an **abstraction** refering to a composite role A based on the roles $R_1, \ldots, R_n$.

Expressions constructed from role names and role constructors are **composite roles**. For example, the value restriction

$$\forall \texttt{ create;!(retrieve+update) } . \texttt{ postState}$$

is based on the composite role `create;!(retrieve+update)`.

Axioms in this description logic allow us to reason about service behaviour. Questions concerning the consistency and role composition with respect to behaviour protocols can be addressed. For instance [13]

$$\forall R\,;S.C \Leftrightarrow \forall R.\forall S.C \quad \text{and} \quad \forall R + S.C \Leftrightarrow \forall R.C \sqcup \forall S.C$$

are two axioms that describe logical properties of the two role combinators sequence (;) and choice (+). The equivalence

$$\forall R.C \sqcap D \Leftrightarrow \forall R.C \sqcap \forall R.D$$

is a pure logical axiom that describes a property of the $\sqcap$-combinator.

A special form of a role constructor are quantified constructors:

- The role expression $\exists(u_1, \ldots, u_n).P$ is an **existential predicate restriction**, if P is an n-ary predicate of a concrete domain – concepts can only be unary – and $u_1, \ldots, u_n$ are role chains.
- Analogously, we define a **universal predicate restriction** $\forall(u_1, \ldots, u_n).P$.

For example, $\exists(x, y).equal$ expresses that there are role values (sometimes called role fillers) for the two roles x and y that are equal. The expression $\forall(x, y).equal$ requires all role values to be equal.

4.2 Names and Parameterisation

The ontology language that we have defined by introducing role constructors for service composition is not yet complete. We can formulate process expressions in terms of service names, but we cannot refer to *data* and we cannot express *service parameters*, cf. Section 2.1.

In ontology languages, individuals are introduced in form of assertions. For instance, `Doc(D)` says that individual `D` is a document `Doc` and `length(D,100)` says that the length of `D` is 100.

- An **individual** x with $C(x)$ is interpreted by $x^I \in \mathcal{S}$ with $x^I \in C^I \subseteq S$.
- The **set constructor**, written $\{a_1, \ldots, a_n\}$ introduces the individual names $a_1, \ldots, a_n$.
- The **role filler** $R : a$ is defined by $(R : a)^I = \{b \in \mathcal{S} | (b, a^I) \in R^I\}$, i.e. the set of objects that have a as a filler for R.

This allows us to introduce individuals on the level of concepts and roles. The fills constructor $R : a$ for a role stands for all objects that have a as a filler of role R.

The essential difference between classical description logic and our variant here is that we need names to occur in role and concept descriptions. A description logic expression $\forall$`create.valid` usually means that `valid` is a concept, or predicate, that can be applied to some individual object; it can be thought of as $\forall$`create(x).valid(x)` for an individual `x`. If roles are services, then `x` should not represent a concrete individual, but rather a name or a variable. For instance the creation service `create` has a parameter `id`. Our objective is to introduce names into the description language. We extend the language defined earlier on by *parameterised roles*.

- We denote a **name** n by a role $\underline{n}_N$, defined by $\underline{n}_N^I = \{(n^I, n^I)\}$.

– A **parameterised role** is a transitional (functional) role R applied to a name $\underline{n}_N$, i.e. $R \circ \underline{n}_N$.

The name definition $\underline{n}_N$ is derived from the role filler and the identity role definition, i.e. $(\underline{n}_N)^I(n^I) = (id : n)^I$.

With names and parameters our *Grid service composition language* is now complete. We can define *process expressions* consisting of services that achieve some defined goals. We can now express a parameterised role

$$\forall \texttt{create} \circ \underline{\texttt{id}}_N.\texttt{post}$$

for our example document storage and access services, defined by

$$\{x | \forall y.(x, y) \in (\texttt{create} \circ \underline{\texttt{id}}_N)^I \rightarrow y \in \texttt{post}^I\}$$

which is equal to $\{\texttt{id}^I | y \in \texttt{post}^I\}$, where y is a *postState* element that could be further described by roles such as $y = \forall \texttt{postCond.post} \sqcap \forall \texttt{outSign.out}$. With names and role composition *parameterised role chains* can be expressed:

$$\forall\ \texttt{update} \circ (\underline{\texttt{id}}_N, \underline{\texttt{doc}}_N); \texttt{postCond . equal(retrieve(id),doc)}$$

The expression $\forall$ `retrieve` $\circ$ $(\underline{\texttt{id}}_N)$ $\circ$ `outSign.(Doc)` is another example[3].

The *tractability of reasoning* about descriptions is a central issue for description logic. The richness of our description logic has some negative implications for the complexity of reasoning. However, some aspects help to reduce the complexity. We can restrict roles to functional roles. Another beneficial factor is that for composite roles negation is not required. The defined language is therefore *decidable*, i.e. the satisfiability problem is decidable. It is based on the language $\mathcal{ALC}$ introduced in [10]. It introduces additionally name and parameterisation constructs based on functional roles, avoiding negation – which preserves the decidability in our extension.

5 Matching of Semantic Grid Service Processes

Dynamic assembly of services, for instance to higher-level services, is a central feature of Grid service architectures. The virtualisation of services through the Web Services Framework enables composition; specific semantic support for matching is, however, desirable.

The *activities* that we are concerned with are *service description, composition* and *matching*. Central *reasoning constructs* of description logics to support these are equivalence and subsumption. In this section, we look at service matching based on interaction protocols and how it relates to subsumption reasoning.

[3] We often drop the $_N$-annotation if it is clear that a name is under consideration.

5.1 Subsumption

Subsumption is the central inference construct in description logics [10]. Subsumption is the *subclass relationship*. It is often used to describe classification hierarchies. **Axioms** based on subsumption and equivalence are introduced into description logics to reason about concept and role descriptions.

- Concepts: subconcept $C1 \sqsubseteq C2$, concept equality $C1 \equiv C2$,
- Roles: subrole $R1 \sqsubseteq R2$, role equality $R1 \equiv R2$,
- Individuals: individual equality $\{x\} \equiv \{y\}$.

The semantics of these axioms is defined based on set inclusion of interpretations for the **subsumption** $\sqsubseteq$ and equality for **equivalence** for $\equiv$. Therefore, $A \equiv B$ iff $A \sqsubseteq B \sqcap B \sqsubseteq A$ is a consequence of the axiom definitions. Subsumption is not implication. Structural subsumption (subclass) is weaker than logical subsumption (implication). Subsumption is defined by subset inclusions for concepts and roles:

- A **subsumption** $C_1 \sqsubseteq C_2$ between two concepts C_1 and C_2 is defined through set inclusion for the interpretations $C_1^I \subseteq C_2^I$.
- A **subsumption** $R_1 \sqsubseteq R_2$ between two roles R_1 and R_2 holds, if $R_1^I \subseteq R_2^I$.

We can embed these axioms into the reasoning framework. For instance $C_1 \sqcap C_2 \sqsubseteq C_1$ or $C_2 \rightarrow C_1$ implies $C_2 \sqsubseteq C_1$ holds for concepts C_1 and C_2. We can use subsumption to reason about matching of two service process descriptions (defined as transitional roles).

5.2 Matching of Service Process Descriptions

A notion of *consistency of composite roles* that define interaction protocols or higher-level services through process expressions relates to the underlying service properties, which could be based on semantical contract-related properties. Often, states or state transitions are constrained through invariants and pre- and postconditions[4].

A concept description $\forall P(R_1, \ldots, R_n).C$ with composite transitional role P is **reachable** if $\{(a, b) \in P^I | \exists b.b \in C^I\}$ is not empty. A composite role $P(R_1, \ldots, R_n)$ is **consistent**, if the last state is reachable.

For instance, in the presence of pre- and postconditions, a composite transitional role P is **consistent** if the following (sufficient) conditions are satisfied:

- for each sequence $R; S$ in P : $\forall postCond.post_R \sqsubseteq \forall preCond.pre_S$
- for each iteration $!R$ in P : $\forall postCond.post_R \sqsubseteq \forall preCond.pre_R$
- for each choice $R + S$ in P : $\forall preCond.pre_R \sqcap \forall preCond.pre_S$ and $\forall postCond.post_R \sqcap \forall postCond.post_S$

[4] Even though we do not fully formalise a framework for pre- and postconditions, we consider these to be of importance for the Semantic Grid [1, 2]. Consequently, we prepare our ontology for future extensions in this direction; see also Section 6.

We can now define *consistent services processes*. A **service process** is a consistent composite role expression $P(R_1, \ldots, R_n)$ constructed from transitional role names $R_1, \ldots, R_n$ and connectors ; , ! , and $+$[5] . The specification of service processes describes the ordering of observable activities of the service process that implements the process expression.

A **protocol transition graph** $G = (N, E)$ for composite transitional roles is a graph that represents all possible process executions. A *transition graph* $G = (N, E)$ can be *constructed inductively* over the syntactical structure of a composite role expression. This transition graph can be related to Kripke transition systems in which we interpret expressions: $N \subseteq S$ is a subset of states; $E \subseteq R$ is a subset of relations for a KTS M with states S and roles R.

The next step is to define *matching* of *consistent service processes*. The matching approach here serves two purposes:

- Does an existing process that realises some goal matches some given requirements? Can this process be selected from a repository?
- What is the relation between two given process expressions? Can one be considered as a refinement of an other?

Process calculi suggest *simulations* and *bisimulations* as constructs to address the *subsumption* and *equivalence* of service processes [15]. We will use a notion of simulation between processes to define service process matching.

- A provider service process $P(S_1, \ldots, S_k)$ **simulates** a requested service process $R(T_1, \ldots, T_l)$, if there exists a homomorphism μ from the transition graph of R to the transition graph of P, i.e. if for each $R_g \xrightarrow{T_i} R_h$ there is a $P_k \xrightarrow{S_j} P_l$ such that $R_g = \mu(P_k)$ and $R_h = \mu(P_l)$.
- We say that a provided service process $P(S_1, \ldots, S_k)$ **matches** a requested service process $R(T_1, \ldots, T_l)$, if $P(S_1, \ldots, S_k)$ simulates $R(T_1, \ldots, T_l)$.

The form of this definition originates from the simulation definition of the π-calculus [15]. The provider needs to be able to simulate the request, i.e. needs to meet the expected behaviour of the requested process. The problem with this definition is that it involves a semantical structure. We can, however, construct a transition graph based on the syntactical representation.

In our document service example, the provider might require the *interaction protocol* `crtDoc;!(rtrDoc+updDoc);delDoc` and the requestor might formulate a *higher-level service* `create;!(retrieve+update)`. Assuming that the operation pairs `crtDoc`/`create`, `rtrDoc`/`retrieve`, and `updDoc`/`update` match based on their contract-relevant descriptions, we can see that the provider matches (i.e. simulates) the required server interaction protocols. `delDoc` is not requested.

We can expect service process *matching* not to be the same as *subsumption*. 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

[5] We often drop service parameters in expressions if only the ordering is relevant.

process expression, there needs to be a corresponding provided service. However, matching is a *sufficient condition* for subsumption.

If service process $P(S_1, \ldots, S_k)$ simulates service process $R(T_1, \ldots, T_l)$, then $R \sqsubseteq P$. If $P(S_1, \ldots, S_k)$ simulates $R(T_1, \ldots, T_l)$, then for each $(a, b) \in R^I$ there is a pair $(a, b) \in P^I$. Therefore, $R^I \subseteq P^I$, and consequently $R \sqsubseteq P$ follow.

6 Semantics – the Wider Picture

Supporting the Semantic Grid [1, 2] is one of your key objectives. In this section, we will briefly address wider implications of semantics in the Grid context. We will also discuss related work in this context.

6.1 The Semantic Web

The Semantic Web initiative [4] bases the formulation of ontologies on two Web technologies for content description: XML and RDF/RDF Schema. RDF Schema is an ontology language providing classes and properties, range and domain notions, and a sub/superclass relationship. Web ontologies can be defined in OWL – an ontology language whose primitives are based on XML and RDF/RDF Schema, which provides a much richer set of description primitives. OWL can be defined in terms of description logics. However, OWL uses a different terminology; corresponding notions are class/concept or property/role.

With the current wide acceptance of the Web and the potential of the Semantic Web as an ontology framework, the combination of Semantic Web and Grids to the Semantic Grid [1] is the obvious choice. We have developed our ontology within this context.

6.2 Semantic Service Contracts

Service level agreements are called contracts. We have already mentioned pre- and postconditions as possible, behaviourally oriented properties that form part of a contract between service provider and service user. For instance, our matching notion depends on a consistency notion capturing these types of descriptions.

We could add pre/postcondition specifications to our basic ontology language – as indicated in Fig. 2, see [16] for details. Then, the formula

$$\forall \texttt{update} \circ \texttt{(id,doc)}.\forall \texttt{postCond}.\texttt{equal(retrieve(id),doc)}$$

in our description logic corresponds to

$$[\texttt{update(id,doc)}][\texttt{postCond}]\ \texttt{retrieve(id)=doc}$$

in a dynamic (modal) logic. Schild [12] points out that some description logics are notational variants of modal logics. This correspondence allows us to integrate modal axioms and inference rules about processes into description logics.

6.3 Semantic Services Ontologies

Some effort has already been made to exploit ontology technology for the software domain [9, 17], mainly for the Web Services Framework [18]. Compositionality has, however, often not been at the centre of these investigations.

OWL-S [9] (aka DAML-S) is an OWL ontology for describing properties and capabilities of Web services. OWL-S represents services as 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 workflows and possible execution paths of a service, i.e. properties that concern the implementation. OWL-S provides to some extend what we aim at for Semantic Grid services. However, our reasoning and ontology support is not possible in OWL-S, since services are modelled as concepts and not rules in the OWL-S ontology. Only considering services as roles makes modal reasoning about process behaviour possible.

7 Conclusions

Grids are services-based infrastructures. In order to make full use of this infrastructure, services need to be composable. Users of a Grid infrastructure need to be able to compose services, i.e. define processes that specify the execution of a composed higher-level service process based on a number of individual services. These processes implement more comprehensive goals and business processes. We have defined a service composition language for Grid services.

Our aim is to support the Semantic Grid – knowledge and semantics are expected to be of importance in the future. Consequently, we have embedded our service composition language into a process-oriented ontological framework that allows the intrinsic description of and reasoning about Semantic Grid services. This ontological framework enables the integration with other semantical aspects, e.g. property descriptions that are relevant for contract formulations in Grid marketplaces. With Grid service technology moving towards Web services, in particular semantic Web service techniques can provide solutions.

We have focused our investigation on an abstract service composition framework, neglecting detailed explorations of different types of concrete services of the individual Grid architecture layers. Addressing these different service types is an issue that we will look at in the future. Equally important is the further study of a variety of Grid application domains. So far, we have combined a case study with experience in other service-oriented architectures.

References

1. D. De Roure, N. Jennings, and N. Shadbolt. The Semantic Grid: A Future e-Science Infrastructure. *International Journal of Concurrency and Computation: Practice and Experience*, 2003.

2. H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-Based Resource Matching in the Grid - The Grid Meets the Semantic Web. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 706–737. Springer-Verlag, LNCS 2870, 2003.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5), May 2001.
4. W3C Semantic Web Activity. Semantic Web Activity Statement, 2002. http://www.w3.org/sw.
5. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: Open Grid Services Architecture for Distribution Systems Integration. In *Proceedings GGF4 Conference, February 2002*. http://www.globus.og/research/papers/ogsa.pdf, 2002.
6. F. Bonnassieux, R. Harakaly, and P. Primet. Automatic Services Discovery, Monitoring and Visualisation of Grid Environments: The MapCenter Approach. In *Proc. European Across Grids Conference 2003*, pages 222–229. Springer-Verlag, LNCS 2970, 2004.
7. W. Poompatanapong and B. Piyatamrong. A Web Service Approach to Grid Information Service. In *Proc. Int. Conference in Web Services ICWS'2003*. 2003.
8. N. Jennings. An Agent-based Approach for Building Complex Software Systems. *Communications of the ACM*, 44(4), 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. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
11. C. Pahl. Components, Contracts and Connectors for the Unified Modelling Language. In *Proc. Symposium Formal Methods Europe 2001, Berlin, Germany*. Springer-Verlag, LNCS-Series, 2001.
12. K. Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. 12th Int. Joint Conference on Artificial Intelligence*. 1991.
13. Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier, 1990.
14. V. Issamy and C. Kloukinas. Automating the Composition of Middleware Configurations. In *15th International Conference on Automated Software Engineering ASE'00*. IEEE, 2000.
15. D. Sangiorgi and D. Walker. *The π-calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001.
16. C. Pahl. An Ontology for Software Component Matching. In *Proc. Fundamental Approaches to Software Engineering FASE'2003*. Springer-Verlag, LNCS Series, 2003.
17. 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.
18. World Wide Web Consortium. *Web Services Framework*. http://www.w3.org/2002/ws, 2003.