Ontology-based Composition and Matching for Dynamic Cloud Service Coordination

Claus Pahl, Veronica Gacitua-Decar, MingXue Wang, Kosala Yapa Bandara

School of Computing
Dublin City University
Dublin, Ireland
E-mail: [cpahl|vgacitua|mwang|kyapa]@computing.dcu.ie

Abstract: Recent cross-organisational software service offerings, such as cloud computing, create higher integration needs. In particular, services are combined through brokers and mediators, solutions to allow individual services to collaborate and their interaction to be coordinated are required. The need to address dynamic management - caused by cloud and on-demand environments - can be addressed through service coordination based on ontology-based composition and matching techniques. Our solution to composition and matching utilises a service coordination space that acts as a passive infrastructure for collaboration where users submit requests that are then selected and taken on by providers. We discuss the information models and the coordination principles of such a collaboration environment in terms of an ontology and its underlying description logics. We provide ontology-based solutions for structural composition of descriptions and matching between requested and provided services.

Keywords: Service coordination; Tuple space; Dynamic service composition; Service Ontology; Cloud Service Coordination; Cloud Mediation.

1 Introduction

Service-oriented architecture (SOA) as a methodological framework aims at providing a service-based infrastructure for interoperable development and integration [1]. However, recent trends such as on-demand computing and service outsourcing [2], which are the main drivers behind cloud computing, pose challenges in terms of the flexibility of composition and also scalability. An emerging architectural solution focuses on brokered or mediated cloud services, where individual cloud services are combined by brokers or end-users and dynamically coordinated by mediator components to satisfy a complex computational need. Dynamic coordination of these services is our focus here.

We introduce an ontology-based solution for service collaboration and coordination, focussing on its description logic foundations, that makes a step from static service architectures (based on Web service orchestration) to dynamic coordination of mediated cloud services [3]. The coordination solution based on a coordination space addresses the need to support dynamic collaboration through semantic matching of providers and requesters at runtime. It enables the self-organisation of service communities through flexible dynamic composition of service architectures.

In contrast to existing mediation solutions, where providers initially publish their services and where clients search for suitable services, here the approach is reversed - changing from a pull-mode to a push-mode where the client posts requests that can be taken on by providers. Different coordination models have been proposed [4, 5, 6]. Domain- and application context-specific solutions [7, 8, 9] and approaches based on semantic extensions are investigated [10, 11], which have also been applied to service composition and mediation. We built up on these semantic mediation approaches by adding a process perspective and by linking this to a coordination technique for requests of services [12]. We focus on the structural composition of objects and processes as part of service requests to support the coordination of requests and provided services. We use an ontology-based formalisation for description and subsumption-based matching. Our contribution is an ontology language for request coordination that adds a process view to existing service matching. We specifically investigate the structural composition of request elements within a dynamic coordination context.

The paper is organised as follows. The next section discusses the context of service collaboration. Section 3 introduces our coordination solution. In Section 4, we address the description of requests and services in the coordination space in terms of ontology-based specification and composition techniques. In Section 5, the matching-based coordination is defined. Section 6

discusses evaluation aspects. We discuss related work in Section 7 before ending with some conclusions.

2 Service Collaboration and Coordination Spaces

Cloud and on-demand computing are emerging as new forms of providing and consuming software as services to enable an integrated collaboration of service communities. Applications often exhibit a more dynamic nature of interaction, particularly, if the service combinations are dynamically brokered and mediated, which requires techniques for the identification of needs and behaviours and the association and customisation of provided services to requested needs [13].

A scenario shall motivate our solution. *Customer care* is a classical enterprise software scenario (layered on top of a full software system) that can be enhanced through distributed, on-demand collaboration infrastructure.

- Sample application objects are software objects, problem descriptions and help files.
- Activities include explanations or activities to repair/adapt software.
- Two sample processes are a software help wizard that guides an end-user through a number of steps to rectify a problem and a customer care workflow that in a number of steps identifies a problem, decides on a resolution strategy and implements the latter (e.g. through adaptation/change of components).

Initially, a user asks for help by posting a request referring to a software component (e.g. a search feature) and a problem (e.g. help file not OK), see Fig. 1. An analysis service takes on the task and determines whether explanation and guidance is sufficient or whether the software itself needs to be changed. In both cases, new requests (objects and goals) are generated. In the first case, the discovery of suitable responses (e.g. by correcting help files) is the aim. In the second case, software changes need to be implemented. Different providers (private cloud service providers in our context) might compete for the same request and only one will be selected based on the dynamic context. Automatically identifying the ongoing process pattern allows a more targeted processing of the initial goal.

The currently most widely adopted approach to service composition is to develop service processes that are orchestrations of individual services [1]. Service orchestrations are executable process specifications based on the invocation of individual services, e.g. in WS-BPEL, the business process execution language. While this is successful for intra-organisational software integration, limitations exist in particular for on-demand and software outsourcing activities such as cloud computing.

- Firstly, the inflexible nature: common to both orchestration and choreography is the static nature of these assemblies, requiring them to be pre-defined, which can only be alleviated to some extent through dynamic adapter generation [15].
- Secondly, the lack of scalability: orchestrations are simple process programs without abstraction mechanisms, thus restricting the possibility to define complex and manageable system specifications.

A look at the scenario illustrates the identified limitations. Increasing flexibility of composition by allowing partners (service users and providers) to dynamically join or leave the community is not possible using the classical approach using hard-coded orchestrations or choreographies. The dynamic requesting and providing of services (by asking for activities to be executed on objects to achieve a goal) avoids complex, pre-defined definitions of service orchestrations or choreographies, thus making the coordination more scalable through self-organisation.

3 Service Coordination Spaces

The solution to address flexibility and scalability of collaboration is a coordination space, which acts as a passive infrastructure to allow communities of independent users (initially requestors) and providers to collaborate through matching of requests and provided services, see Fig. 2. It is governed by coordination principles:

- tasks to perform an activity on an object occur in states
- services collaborate and coordinate their activities to execute these tasks
- advanced states are reached if the execution is permitted by guards

The central concepts are objects and goals (the latter reflecting outcomes of activities) provided together as services and processes that are seen as goal-oriented assemblies of services. Service requesters enter a typed object together with a goal that defines the processing request. Service and process providers can then select (match) this processing request. The coordination space is complemented by a knowledge space, which provides the crucial reasoning support for matching.

The coordination space is a repository for requests that allows requests and potential providers to be coordinated. The knowledge space provides platform functionality, e.g. the request matching and required transformations. The Web service platform provides with UDDI a static mediator in repository form that is not suitable to support dynamic collaboration. Our proposal is also different from UDDI in a more fundamental

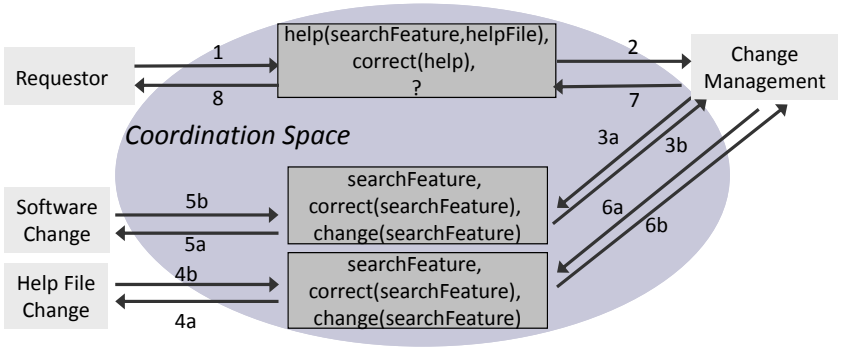

Figure 1 Coordinated Process of Requests.

way. It changes the perspective of the client from a pull- to a push-approach. Instead of querying (pull) the repository for suitable published entries, they submit (push) requests here, which in turn are picked up by providers.

4 Request Specification, Transformation and Composition

Based on an information model for coordination requests and its underlying ontological foundations, three aspects shall be investigated:

- the specification of coordination requests using the ontology-based information model – this forms the foundations for semantic matching, since requests are at the core of the coordination approach,
- the transformation between different object representations to allow adaptation between coordination participants to allow for an increased degree of flexibility for inexact request matches,
- the composition of request elements to facilitate request decomposition and the enablement of a process-style request processing.

From an ontology technology perspective, we need to define a request language (with support for composed objects and processes through composition operators and role expressions), an ontology mapping technique (for adaptation purposes), and a formalisation of composition for the three aspects, respectively. We start with an information model and ontology basics, before addressing the three aspects in turn.

4.1 Information Model

Users are usually concerned with processing objects such as electronic documents passing through business processes. The central concepts of our *information model* are objects, goals and processes, which together form *requests*:

- Changing, evolving *objects* are dynamic entities. This follows trends to focus on structured objects and documents as the central entities of processing, as proposed by ebXML and other business standards.
- *Goals* are declaratively specified, expressing the requested result of processed objects [16]. Essentially, the aim is to allow users and providers to refer to them declaratively, e.g. in the form of a goal-oriented user request (requesting object processing) and to enable semantic goal-based matching.
- The *process* notion refers to business and workflow processes. States of the process are points of variation for objects: object data evolves as it passes through a process. Goals relating to objects are expressed in terms of states of the processes where a process state is considered at the level of individual object modifications. The link to objects is provided via states of processes. Process-centricity is the central feature of service coordination here, allowing us to retain the compositional principle of Web services.

Cloud computing as an information processing and management infrastructure technology would benefit from requests being formulated in terms of the underlying information objects being processed, an abstract specification of goals and the process that the individual processing activities are embedded in. We will now formalise this information model in terms of a description logic-based ontology for the specification and composition of requests.

4.2 Ontologies and Description Logic

Our solution is a description logic-based composition ontology to support matching between requests and provided services. Ontologies are a good candidate for the semantic, goal-oriented specification of objects and processes [17]. We introduce the core of the description

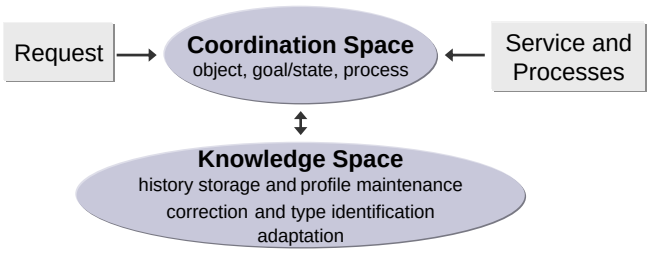

Figure 2 Coordination and Knowledge Space Architecture.

logic language $\mathcal{ALC}$ [18], which defines ontology languages like OWL-DL. $\mathcal{ALC}$ provides combinators and logical operators that suffice for our service composition ontology. It consists of three basic elements:

- *Concepts* are the central entities. Concepts are classes of objects with the same properties. Concepts represent sets of objects.
- *Roles* are relations between concepts. Roles define a concept through other concepts. We distinguish two role types: *descriptive* roles to define static properties and *transitional* roles to define activities (object state changes in processes).
- *Individuals* are named objects.

Individuals can be thought of as constants, concepts as unary predicates, and roles as binary predicates. A Tarski-style model semantics based on an interpretation I maps concepts and roles to corresponding sets and relations, and individuals to set elements. Properties are specified as *concept descriptions*:

- *Basic concept descriptions* are formed as follows: A denotes an atomic concept; if C and D are (atomic or composite) concepts, then so are $\neg C$ (negation), $C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), and $C \rightarrow D$ (implication).
- Value restriction and existential quantification, based on roles, are concept descriptions that extend the set of basic 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.

The combinators are defined using classical set-theoretic, i.e. extensional concept interpretations. Given a value set $\mathcal{S}$, we define the model-based *semantics* of *concept descriptions* as

$$\top^I = \mathcal{S} \text{ and } \bot^I = \emptyset$$
$$(\neg A)^I = \mathcal{S} \backslash A^I \text{ and } (C \sqcap D)^I = C^I \cap D^I$$
$$(\forall R.C)^I = \{a \in S \mid \forall b \in S.(a,b) \in R^I \rightarrow b \in C^I\}$$
$$(\exists R.C)^I = \{a \in S \mid \exists b \in S.(a,b) \in R^I \wedge b \in C^I\}$$

Combinators $\sqcap$ and $\rightarrow$ can be defined based on $\sqcup$ and $\neg$ as usual. An *individual* x defined by $C(x)$ is interpreted by $x^I \in \mathcal{S}$ with $x^I \in C^I$.

Structural subsumption $\sqsubseteq$, used for service component matching in Section 5, is a relationship defined by subset inclusions for concepts and roles. Structural subsumption (subclass) is weaker than logical subsumption (implication) [18]. Subsumption can be further characterised by axioms such as the following for concepts C_1 and C_2: $C_1 \sqcap C_2 \sqsubseteq C_1$ or $C_2 \rightarrow C_1$ implies $C_2 \sqsubseteq C_1$. The expression $C_1 \equiv C_2$ means equality. The concept descriptions can be mapped to predicate logic, which clarifies the reasoning capabilities of the approach. A concept C can be thought of as a unary predicate $C(x)$ for a variable x and roles R as binary predicates $R(x,y)$, i.e. concept descriptions like $\exists R.C$ are mapped to $\exists y.R(x,y) \wedge C(x)$.

4.3 Ontology-based Specification of Requests

We semantically enrich an information model – a conceptualisation – capturing object structure, object modification states and an object evolution process [19]. Ontologies with descriptive and operational layers through an encoding of dynamic logic in a description logic provide the foundations for our object and process specification framework. This allows us include behavioural and temporal aspects into the core ontology framework capturing objects [20][1].

- Objects types are represented as concepts in a domain ontology. Objects in the form of XML data schemas, embedded into an assumed domain ontology, represent the object type. A composition relationship becomes a core generic relationship for our request ontology (in addition to the traditional subsumption-based taxonomic relationship). Structural nesting of XML elements is converted into ontological composition relationships.

  Sample objects are a *searchFeature* and a *helpFile*, connected by a *help* role.
- Goals are properties of the object concepts stemming from the ontology (covering domain-specific properties as well as software qualities). Goals are expressed in terms of ontology-based properties of objects (concept descriptions), denoting states of an object reached through processing.

A sample goal is *correct(help)*, which might need to be resolved by modifying the respective role source and target.

- Processes are based on a service process ontology with service process composition operators in the form of transitional roles (like sequencing ';', choice '+' or iteration '!' – see [20] for details) as part of the ontology. Processes are specified based on input and output states linked to goals as properties.

  A sample process is *analyse(help); (change(searchFeature) + change(helpFile))*.

A sample object is a software component, the goal the request to change parameters. This would form a semantically annotated service goal specification

$$\exists change.typeOf(cmp, TypeA) \sqcup typeOf(cmp, TypeB)$$

here requesting the object to be changed such that the type of the component *cmp* is one of the specified $TypeA$ or $TypeB$. *change* is a transitional role here.

A software component as an object in the context of customer care has properties such as deployed, analysed, or changed. A maintenance process could be expressed as a cyclic process $!(deploy; analyse; change)$ which defines an iteration of the 3-sequence of activities. In terms of the ontology, this process specification is a composed role description that can be further specified, e.g.

$$\forall(deploy; analyse; change).consistent(state)$$

saying that the sequence is expected to result in a consistent state. We call these roles *transitional* as they result in state transitions. A subprocess has been used above in the process part of the sample request triple.

The notions of a request specification and its semantics need to be made more precise. We assume a request to be a specification $request = \langle \Sigma, \Phi \rangle$ based on the elementary type ontology with a signature $\Sigma = \langle C, R \rangle$ consisting of concepts C and roles R and concept descriptions $\phi \in \Phi$ based on Σ which cover both goals and processes through descriptive and transitional roles. A *request* is interpreted by a set of models M. The model notion refers to algebraic structures that satisfy all concept descriptions ϕ in Φ. The set M contains algebraic structures $m \in M$ with classes of elements C^I for each concept C, relations $R^I \subseteq C_i^I \times C_j^I$ for all roles $R : C_i \rightarrow C_j$ such that m satisfies the concept description. Satisfaction is defined inductively over the connectors of the description logic $\mathcal{ALC}$ as usual [18]. A signature Σ defines the request language vocabulary, e.g. consisting of domain-specific object *component*, activities *change* or *deploy*, and property *typeOf*.

The combination of two request specifications should be conflict-free, i.e. semantically, no contradictions should occur. A *consistency* condition can be verified by ensuring that the set-theoretic interpretations of two objects S_1 and S_2 are not disjoint, $S_1^I \cap S_2^I \neq \emptyset$, i.e. their combination is satisfiable and no contradictions occur.

4.4 Object Adaptation and Transformation

Objects are represented through XML schemas that in practice implement the signature construct Σ from the previous section. Often, particularly in *multi-tenancy* situations where a service is shared, the user representation of objects will not always that of the provider (e.g. when for instance user organise and access help files in different ways locally) and, consequently *transformations* are necessary to *adapt structural representation* differences. The consequence are inexact matches due to syntactical representation differences. This can be remedied by transformation rules based on a domain ontology that map between semantically equivalent, but structurally different representations. In the given scenario, this ontology would formalise software concepts. Here, the internal structure of an XML-based help file might vary, but cover the same aspects.

Domain ontologies define a knowledge-based object information model to constrain the integration and coordination model and make the transformation rules for the adaptation between representations consistent with the object models. This enhances the basic XML-based object schema representation to an ontology-based model on which ontology-based mappings can be defined.

4.4.1 Ontology-based Semantic Information Model

In order to avoid the verbosity of the XML-based OWL, an ontology-based representation akin to the Manchester syntax for OWL is used. A software and help data type that characterises the object structures for the case study can be semantically defined:

SoftwareFeature =
  ∃ *componentID* . *Identification* ∧
  ∃ *featureName* . *Name* ∧
  ∃ *provServices* . *Service*
Help =
  ∃ *helpfileID* . *ID* ∧
  ∃ *suppFeature* . *SoftwareFeature*

Each object, like *Help*, is defined in terms of its properties, such as *helpfileID*. These properties associate information of a specific type or another concept to a given concept - *helpfileID* assigns an *Identification*, *suppFeature* another concept called *SoftwareFeature*. This description defines a semantic information model, represented graphically in Fig. 3, following Protégé in distinguishing data and object properties.

4.4.2 Schema Generation

This ontology model opens the opportunity to automatically generate coordination components, i.e. data schemas and transformation rules. Both are central ingredients for the coordination management. A *canonical XML object schema* can be automatically derived from this model, as the following example demonstrates:

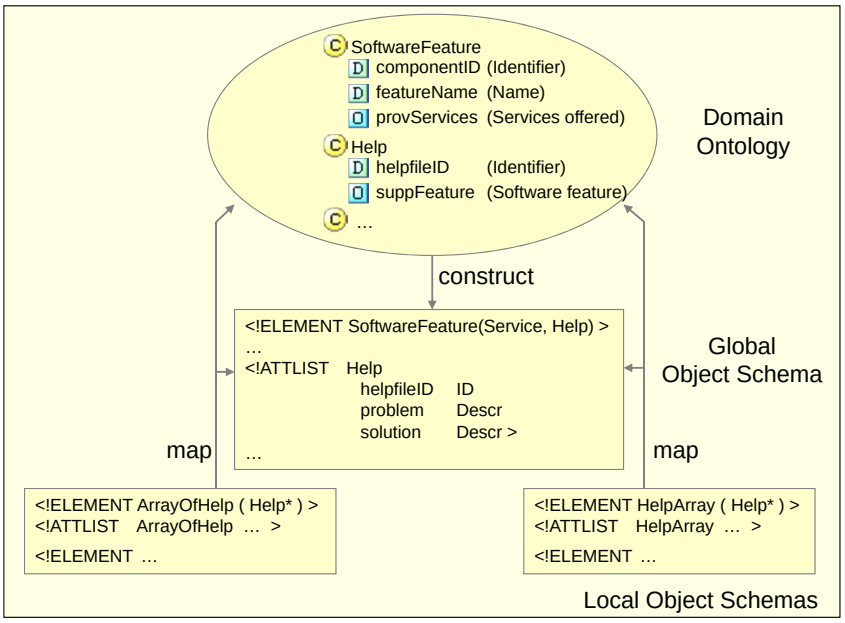

Figure 3 Semantic Information Model for the Feature and Help Objects.

```
<!ELEMENT SoftwareFeature(Service, Help) >
```

is a the element definition for a concept and properties can be represented as attributes:

```
<!ATTLIST Help
      helpfileID    ID
      problem       Descr
      solution      Descr >
```

The *global object integration schema* is now defined as an abstract and interchangeable domain model. Any *local object schemas* are defined by mappings into either a global schema or directly into the domain ontology, see Fig. 3.

4.4.3 Transformation Rule Generation

In addition to the canonical object schemas, more importantly *transformations between local schemas* can also be derived automatically from the ontology model and the schema mappings. A prerequisite is that each of the local schemas is mapped to the domain ontology.

While a basic solution for this transformation can easily be generated based on the semantic integration of all data aspects, an adequate transformation generation should also consider beneficial properties such as modularity. An algorithm for *rule construction* from the information model to an integration feature in the coordination space can be defined that creates modular ontology mapping rules - modular in the sense of there being a rule for each concept:

1. define a *construction rule* per concept of the target schema, based on concepts from the ontology. We define an ontology mapping τ based on an ontology graph $O = (C, R)$ as $\tau_c(P_1, \ldots, P_n) = T$ with $T = (C', R')$ where the P_i are subgraphs of O and $c \in C$ (such as *helpArray*). O and T are local object schemata (ontologies).

2. identify *semantically equivalent concepts* in the source schema based on the ontology - local elements map to the same ontology concept. For each $c' \in C'$ there is a $c \in C$ that is semantically equivalent (i.e. both are mapped to the same concept d in the domain ontology (see Fig. 3).

3. for each concept, determine *attributes* and copy their counterparts from the source schema - which preserves concept properties. For each $r \in R$, there is a role expression, defined based on $r_1, \ldots, r_n \in R$ to allow the combination of roles into possibly nested structures (e.g. object-valued role sequences).

This defines consistent, i.e. semantics-preserving transformations that allows structural adaptation of object representations due to the semantic and not only syntactic integration of data [22]. Applied to the structural formulation of the object schemas of customer and provider, this means that for example the `HelpArray` and a `Help` file transformation rule can be automatically generated.

A sample set of generated transformation rules, here for the XML transformation language `Xcerpt` [14], is given below. The first rule produces the `HelpArray` by grouping and reconstructing individual and sequences of roles.

```
CONSTRUCT
   HelpArray [
      var help,
      var helpfileID,
      all suppFeature [
         var featureName,
         all Service [ var serviceId, var serviceImpl ]
      ]]
   ]]
 FROM
   Help [[ var help, var helpfileID ]]
 AND
   SoftwareFeature [[
      var feature [[
          var service [[
      var serviceImpl ]] ]] ]]
```

Similar to other XML transformation languages, Xcerpt uses a pattern matching approach, using variables to identify structures in the input. The next rule gets `Help` data terms from the original input `ArrayOfHelp`.

```
CONSTRUCT
   Help [ var help, var helpfileID ]
FROM
   ArrayOfHelp [ var help, var helpfileID ]
```

While we have used Xcerpt, any other language for XML transformations can in principle be used. Xcerpt supports the generation of modular rules.

4.5 Ontology-based Object, Goal and Process Composition

The core format for request specifications has been defined, and we have shown how object transformations can be used to make matching more flexible and allow adaptation. We now add support for the compositionality of the request elements, which extends approaches such as [17, 21]. This facilitates the decomposition of requests into processing smaller objects through a composed process of individual object processing activities, as we already illustrated with the *help* decomposition into two smaller steps, see Fig. 1. Thus, the third request element, processes, are included as well.

4.5.1 Composition Principles

Subsumption is the central relationship in ontology languages, which allows concept taxonomies to be defined in terms of subtype or specialisation relationships. In conceptual modelling, composition is another fundamental relationship that focuses on the part-whole relationship. In ontology languages, composition is less often used [18]. The notion of composition can be applied in different ways:

- *Structural composition.* Structural hierarchies of architectural elements define the core of architectures. It can be applied to request objects here.
- *Sequential (and behavioural) composition.* Dynamic elements (processes) can be composed to represent sequential behaviour. Sequential composition can be extended by adding behavioural composition operators like choice or iteration.

We use the symbol "$\rhd$" to express the composition relationship. It is syntactically used in the same way as subsumption "$\sqsubseteq$" to relate concept descriptions.

- *Composition object hierarchies* shall consist of unordered subcomponents, expressed using the component composition operator "$\rhd$". An example is $ProblemDescr \rhd FaultCause$, i.e. a $ProblemDescr$ consists of $FaultCause$ as a part. Composed objects are interpreted by unordered multisets.
- *Processes* can be *sequences* or *complex behaviours* that consist of ordered process elements, again expressed using the composition operator "$\rhd$". An example is $maintenance \rhd analysis$, meaning that *maintenance* is actually a composite process, which contains for instance an *analysis* activity. A more complex decomposed subprocess is $maintenance \rhd analysis; change; deployment$. We see composite process implementations as being interpreted as ordered tuples providing a notion of sequence. For more complex behavioural compositions, graphs serve as models to interpret this behaviour.

4.5.2 Request Composition

We introduce two basic syntactic composition constructs for object and process composition[2], before looking at behavioural composition as an extension of sequential composition:

- The *structural composition* between C and D is defined through $C \rhd \{D\}$, i.e. C is structurally composed of D if $type(C) = type(D) = Object$.
- The *sequential composition* between C and D is defined through $C \rhd [D]$, i.e. C is sequentially composed of D if $type(C) = type(D) = Process$.

Note, that the composition operators are specific to the respective request element. This basic format that distinguishes between the two composition types shall be complemented by a variant that allows several parts to be associated to an element in one expression.

- The structural composition $C \rhd \{D_1, \ldots, D_n\}$ is defined by $C \rhd \{D_1\} \sqcap \ldots \sqcap C \rhd \{D_n\}$. The parts $D_i, i = (1, .., n)$ are not assumed to be ordered.
- The sequential composition $C \rhd [D_1, \ldots, D_n]$ is defined by $C \rhd [D_1] \sqcap \ldots \sqcap C \rhd [D_n]$. The parts D_i with $i = (1, .., n)$ are assumed to be ordered with $D_1 \leq \ldots \leq D_i \leq \ldots \leq D_n$ prescribing an execution ordering $\leq$ on the D_i.

The latter allows us to write $maintenance\_step \rhd [deploy, analyse, redevelop]$ as a composed behavioural specification, which gives semantics to the expression $deploy; analyse; redevelop$.

The semantics of the two composition operators shall now be formalised. So far, models $m \in M$ are algebraic structures consisting of sets of elements C^I for each concept C in the service object signature and relations $R^I \subseteq C^I \times C^I$ for roles R. We now consider elements to be composite:

- Structurally composite concepts $C \rhd \{D_1, \ldots, D_n\}$ are interpreted as multisets $C^I = \{\{D_1^{I^1}, \ldots, D_1^{I^k}, \ldots, D_n^{I^1}, \ldots, D_n^{I^l}\}\}$. We allow multiple occurrences for each concept $D_i, (i = 1, .., n)$. With $c \in C^I$ we denote set membership.
- Sequentially composite concepts $C \rhd [D_1, \ldots, D_n]$ are interpreted as tuples $C^I = [D_1^I, \ldots, D_n^I]$. Tuples are ordered collections of sequenced elements. Apart from membership, we assume index-based access to the tuples in the form $C^I(i) = D_i^I, (i = 1, .., n)$, selecting the i-th element in the tuple.

While subsumption as a relationship is defined through subset inclusion, composition relationships are defined through membership in collections (multisets for structural composition and ordered tuples for sequential composition).

4.5.3 Behavioural Composition

Behavioural specification is based on the process composition operators. These operators allow us to refine a process and specify detailed behaviour. While a basic form of behaviour (sequencing) has been defined, we can extend it to a more comprehensive approach that requires a more complex model semantics (graphs). This has reasoning implications, as we will discuss at the end of Section 5. We define a process P through a behavioural specification: $P \rhd [B]$ where B is a behavioural expression consisting of a basic process P or

- a unary operator '!' applied to a behavioural expression $!B$ (*iteration*), or
- a binary operator '+' applied to two behavioural expressions $B_1 + B_2$, expressing *non-deterministic choice*, or
- a binary operator ';' applied to two behavioural expressions $B_1 \ ; \ B_2$, expressing the previously introduced *sequencing*.

In line with the basic forms of composition, the iteration $P \rhd [!B]$ is defined by $P \rhd [B, \ldots, B]$, the choice $P \rhd [B_1 + B_2]$ is defined by $P \rhd [B_1] \sqcup C \rhd [B_2]$, and the sequence $C \rhd [B_1 \ ; \ B_2]$ is defined as above.

We need to extend the semantic model by interpreting behaviourally composite processes through graphs (N, E) where processes are represented by edges $e \in E$ and nodes $n \in N$ represent connection points for sequence, choice and iteration. The three operators are defined through simple graphs.

5 A Coordination Architecture for Services

The coordination functionality follows established coordination approaches like tuple spaces [4, 5] by providing deposit and retrieval functions for the space elements – tuples which consist of object type, goal and supporting process. Specifically, one deposit and two retrieval functions for the ontology-defined requests from the previous section are provided. We follow here a proposal from [5, 6] that extends the original Linda *out*, *read* and *in* operations:

- *deposit(object!, goal!, process!)*, where the parameters are defined as above in Section 4, is used by the client who deposits a copy of the tuple *[object, goal, process]* into the coordination space. The exclamation mark '!' indicates that values are deposited. The process element is optional; it can be determined later to guide individual processing activities.
- *meet(object?, goal?)*, with parameters as above, is used by a service provider and identifies a matching tuple in the coordination space. The question mark indicates that abstract patterns are provided that need to be matched by concrete deposited tuples. Ontological subsumption reasoning along the object concept hierarchy enhances the flexibility of retrieval by allowing subconcepts to be considered as suitable matches. A subconcept is here defined as a subclass of the concept in terms of the domain ontology, but it also takes the composition properties (see structural composition in the previous section) into account, i.e. requires structural equivalence.
- *fetch(object?, goal?)* is used as *meet*, but it does remove the tuple, blocking further access to the tuple. The coordination space will select the tuple that is deemed to be the closest subsumption-based match, i.e. conceptually the closest in the ontological hierarchy of a given central domain ontology.

meet is used to inspect the tuple space; *fetch* is used if a requested task is taken on and is taken as a commitment to achieve the goal. We assume for simplicity here that provider results are directly communicated to the requester.

Matching in the *meet* and *fetch* operations is the critical activity here and shall be defined in terms of the three components of the ontological framework from the previous section:

- request matching based on a subsumption relationship

- enhanced matching and object adaptation based on schema transformations
- decomposition of requests into internal processes.

We look at the application of the earlier ontology foundations now one-by-one.

5.1 Meet and Fetch - Request Matching.

Subsumption is a relationship defined by subset inclusions for concepts and roles, which is here used to support matching between of provider capabilities and user goals. Providers inspect requests and match them against their own capabilities. We assume an exact match between structural objects here (an extension was introduced in the adaptation and transformation section). Goals are concept descriptions that are *matched* based on a *subsumption* relation. Subsumption is defined as follows:

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

We use subsumption to reason about matching of two request descriptions based on transitional roles. OWL-based subsumption reasoners can support this task.

5.2 Meet and Fetch - Adaptation

The adaptation techniques presented earlier provide a means to enhance the matching. The core version assumes equality of object schemas, which can be relaxed into a *bijective mapping* with the *transformation technique*. If a mapping can be generated using the technique, then two structurally different, but semantically equivalent representation can be matched. The ontology provides the verifiable definition of semantic equivalence here.

5.3 Meet and Fetch - Decomposition into Processes

In Fig. 4, a process emerges from the sequence of events, indicated through numbered coordination operations. If this process is initially not given by the requester, then a process mining tool might identify one to guide and constrain further processing, but that is beyond the scope here. Processes can also be used to structure the cloud negotiation process - which will be discussed later on. The schematic example in Fig. 4 follows the customer care scenario and abstracts its activities that we outlined in Section 2: 1) client deposits the help request (problem description object with guidance as the goal), 2) one service provider meets and fetches the request, 3) provider creates and deposits two more requests - one to create a suitable help file for the problem, the other to determine whether the software needs to be modified (software entity as object and analysis request as goal), 4) these tuples are in turn fetched by other providers, 5) these providers then deposit solutions, 6) which are fetched by the initial provider, 7) who in turn deposits an overall solution, 8) which is finally used by the requester.

6 Evaluation

Our key concern here was the definition of an ontology-based language for dynamic request coordination. The example illustrates the need for the novel solution components of our approach:

- the advanced matching needs, here based on transformation-based object mappings and subsumption-based goal matching,
- the process aspect and the request composition mechanisms, which allow complex tasks to be broken up and managed by a specific process.

We have explored ontology-based solutions for these problems. These entail some theoretical concerns. While description and reasoning capabilities of our ontology solution have been illustrated, the tractability of reasoning is a central issue in the dynamic context here. While the richness of our description logic with complex roles that represent processes has some potentially negative implications for the complexity of reasoning, the complexity can be reduced here. We can restrict roles to functional roles. Another beneficial factor is that for roles negation is not required [18]. Then, decidability is achieved, which is critical for dynamic reasoning. A crucial problem is the decidability of the specification if concrete domains are added. Admissible domains guarantee decidability [18]. A domain D is called *admissible* if the set of predicate names is closed under negation, i.e. for any n-ary predicate P there is a predicate Q such that $Q^D = (S^D)^n \backslash P^D$, there is a name $\top_D$ for S^D, and the satisfiability problem is decidable; i.e. there exists an assignment of elements of S^D to variables such that the conjunction $\wedge_{i=1}^{k} P_i(x_1^{(i)}, \ldots, x_{n_i}^{(i)})$ of predicates P_i becomes true in D. Thus, we can choose concrete domains are admissible [20] based on common conceptualisations of application domains like customer care, e.g. for object descriptions based on the given vocabulary (signature).

After discussing some theoretical concerns, we briefly address the concrete implementation to demonstrate the feasibility of the implementation, which also looks at tractability and performance concerns. The functionality of the coordination and knowledge spaces is currently implemented in the form of infrastructure services. These services are based on Java implementations exposed as services. We use the Jena engine `jena.sourceforge.net` to facilitate the semantic

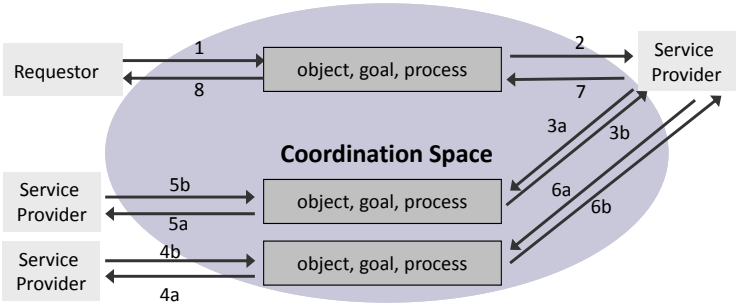

Figure 4 Abstract Coordinated Process of Requests

functions (knowledge space) on top of the LighTS-based coordination space (`lights.sourceforge.net/`). Full behavioural composition is currently not supported, however, this architecture demonstrates the feasibility of ontology-based processing at runtime.

Our results to-date show an acceptable performance overhead of around normally not more than 10% for coordination activities (dynamic matching) in a distributed environment for simple scenarios compared a traditional hardwired WS-BPEL composition of services. The required flexibility gain is achieved (in comparison to a WS-BPEL solution) by enabling dynamic composition.

Besides flexibility, scalability is another issue. Our proposal addresses a scalability problem often associated with more static forms of service composition such as orchestration and choreography. A central success criterion is here the ability to deal with (i.e. match) requests in adequate time (assuming suitable providers). Our results demonstrate a non-linear (polynomial) increase of processing times if we increase input parameters in different dimensions. These dimensions include

- the number of participants (processes that access the tuple space),
- the length of tuples for matching (representing complex requests) and
- the number of different types of requests (representing different service types being handled concurrently).

We have run tests with up to 10000 requestors (with an average of 3 providers per request) and request tuples of up to 10000 bytes. The worst case was a matching time (including deposit, meet to determine the semantic match and fetch to commit) of 10.8 seconds (for the combination of maximum values on a standard PC). In practice, this albeit rare situation can be alleviated by clustering request types and providing targetted coordination spaces for specific (sub)domains, in which case the execution time would reduced by at least an order of magnitude.

7 Related Work

The coordination paradigm applied here is a fundamental change to existing service discovery and matching approaches. Coordination models have been widely used to organise collaboration. The Linda tuple space coordination model [4] has influenced a wide range of variations including our on work on concurrent task coordination in programming languages [5], which in turn has influenced the solution here. More recently, domain- and application context-specific solutions and approaches based on semantic extensions have been investigated [10]. However, dynamic environments have not yet been addressed. Over the past years, coordination has received much attention [7, 8, 9] due to the emergence of collaborative ICT-supported environments, ranging from workflow and collaborative work to technical platforms such as service collaboration. The latter ones have also been applied to service composition and mediation. In [23], an ontology-based collaboration approach is described that is similar to our in that it advocates a push-service object of coordination. We have added to semantic mediation approaches like [10, 23] by including a process notion as a central component of request tuples, supported by a process-centric ontology language. Through the goal/state link, this process context is linked to the request coordination technique focussing on objects are primary entities.

WSMO [17] is an example of a service ontology that enables composition and matching support. Service ontologies are ontologies to describe Web services, aiming to support their semantics-based and, as here, goal-based discovery in Web service registries. The Web Service Process Ontology WSPO [20, 24] is also a service ontology, but its focus is the support of description and reasoning about service composition and service-based architectural configuration, which forms the foundation of the process-element of requests here. We combine here a process-centric ontology (based on WSPO) with composition and deploy this in a dynamic composition environment.

8 Conclusions

Integration and coordination of services is at the core of dynamic service architectures such as cloud computing platforms. Manually designed service applications support software systems in classical sectors such as finance and telecommunications. However, their structural inflexibility makes changes difficult. Current service computing platforms suffer also from scalability problems. Our coordination space techniques enhances collaboration capabilities in the context of dynamic applications. Decoupling requesters from providers through the space achieves flexibility, which particularly benefits the cross-organisational setting of cloud computing. This flexibility is needed to support brokered and mediated cloud service compositions. Scalability can also be achieved through a passive coordination architecture with reduced coordination support - which, however, necessitates the cooperation of providers to engage and pro-actively use the coordination space as a market place.

Our focus here has been on an ontology language for service request composition and matching, which is at the core of an ontological framework for cloud service coordination, particularly addressing the adaptation and (de)composition needs of cloud mediation. Based on a description logic formalisation, it provides composition-oriented description operators and a subsumption-based matching construct. We have specifically looked at tractability problems, which are important for dynamic environments. The main contribution is an ontology-based matching solution that is based on structured, composed requests and the customisation of this framework for a dynamic composition environment. Abstract specifications of data and behaviour, formalised as ontology-based models, are at the core. These models are processed to generate or control service execution. A key concern observed is a trade-off between the richness of the language for matching and performance requirements in dynamic environments.

While we have defined the core coordination principles here, the range of supporting features needs to be investigated further. Part of this are fault-tolerance features supporting self-management and semantic techniques deducing object and process types from possibly incomplete information [27]. Trust is a related aspect that needs to be addressed. We have occasionally indicated advanced functionality; this could further include the automated identification of processes based on stored process history or the possibility to consider non-functional aspects reflected in profiles and context models during matching. The rationale behind the process element in the request triples is to support common interactions, such as negotiation protocols that complete the necessary handshake where the requestor actually approves a provider before proceeding. The interaction sequences can be added by the broker as process patterns into request executions.

References

[1] Alonso, G., Casati, F., Kuno, H. and Machiraju, V. (2004) Web Services - Concepts, Architectures and Applications. Springer-Verlag, Berlin.

[2] Hayes, B. (2008) Cloud computing. Communications of the ACM 51(7):9-11.

[3] Rao, J. and Su, X. (2004) A Survey of Automated Web Service Composition Methods. In Intl. Workshop on Semantic Web Services and Web Process Composition 2004. Springer-Verlag, LNCS 3387, pp. 43-54.

[4] Gelernter, D. (1985) Generative Communication in Linda. ACM Transactions on Programming Languages and Systems 7(1):80-112.

[5] Doberkat, E.-E., Hasselbring, W. and Pahl, C. (1996) Investigating Strategies for Cooperative Planning of Independent Agents through Prototype Evaluation. In Proc. First International Conference on Coordination Models and Languages. Springer-Verlag, LNCS 1061, pp. 416-419.

[6] Doberkat, E.-E., Franke, W., Gutenbeil, U., Hasselbring, W., Lammers, U. and Pahl, C. (1992) PROSET A Language for Prototyping with Sets. In Proc. Third International Workshop on Rapid System Prototyping. pp. 235248.

[7] Johanson, B. and Fox, A. (2004) Extending Tuplespaces for Coordination in Interactive Workspaces. Journal of Systems and Software 69(3):243-266.

[8] Li, Z. and Parashar, M. (2005) Comet: A Scalable Coordination Space for Decentralized Distributed Environments. In Proceedings of the Second international Workshop on Hot Topics in Peer-To-Peer Systems HOT-P2P. IEEE, pp. 104-112.

[9] Balzarotti, D., Costa, P. and Picco, G.P. (2007) The LighTS tuple space framework and its customization for context-aware applications. Web Intelligence and Agent Systems 5(2): 215-231.

[10] Nixon, L., Antonechko, O. and Tolksdorf, R. (2007) Towards Semantic tuplespace computing: the Semantic web spaces system. In Proceedings of the 2007 ACM Symposium on Applied Computing SAC '07. ACM, pp. 360-365.

[11] NIST. Process Specification Language (PSL) Ontology - Current Theories and Extensions. National Institute of Standards and Technology, USA. Available from: http://www.mel.nist.gov/psl/ontology.html [Accessed 10 Oct 2011].

[12] Pahl, C. (2010) Dynamic Adaptive Service Architecture - Towards Coordinated Service Composition. In European Conference on Software Architecture ECSA'2010. Springer-Verlag, LNCS, pp. 472-475.

[13] Pahl, C. and Zhu, Y. (2006) A Semantical Framework for the Orchestration and Choreography of Web Services. In Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005). Electronic Notes in Theoretical Computer Science 151(2):3-18.

[14] Schaffert, S. (2004) Xcerpt: A Rule-Based Query and Transformation Language for the Web. PhD Thesis, University of Munich.

[15] Brogi, A. and Popescu, R. (2006) Automated Generation of BPEL Adapters. In Proc. ICSOC06. Springer-Verlag, LNCS 4294. pp. 27-39.

[16] Andersson, B., Bider, I., Johannesson, P. and Perjons, E. (2005) Towards a formal definition of goal-oriented business process patterns. BPM Journal 11:650-662. 2005.

[17] Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C. and Fensel, D. (2005) Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106.

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

[19] Pahl, C. (2002) A Formal Composition and Interaction Model for a Web Component Platform. In ICALP'2002 Workshop on Formal Methods and Component Interaction. Electronic Notes on Computer Science ENTCS, 66(4).

[20] Pahl, C. (2007) An Ontology for Software Component Description and Matching. International Journal on Software Tools for Technology Transfer 9(2): 169-178.

[21] Arroyo, A. and Sicilia, M.-A. (2008) SOPHIE: Use case and evaluation. Information and Software Technology. 50(12):1266–1280.

[22] Pahl, C. (2007) Semantic Model-Driven Architecting of Service-based Software Systems. Information and Software Technology. 49(8): 838-850.

[23] Tsai, W.T., Xiao, B., Chen, Y. and Paul, R.A. (2006) Consumer-Centric Service-Oriented Architecture: A New Approach. In Proceedings IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and Workshop on Collaborative Computing, Integration, and Assurance. pp. 175-180.

[24] Pahl, C. (2005) Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture. European Conference on Model-Driven Architecture - Foundations and Applications ECMDA'2005. Springer-Verlag, LNCS 3748. pp. 88-10.

[25] Semantic Web Services Language (SWSL) Committee (2006) *Semantic Web Services Framework (SWSF)*. Available from: http://www.daml.org/services/swsf/1.0/ [Accessed 10 Oct 2011].

[26] DAML-S Coalition (2002) DAML-S: Web Services Description for the Semantic Web. In Proc. First International Semantic Web Conference ISWC 2002. Springer-Verlag, LNCS 2342, pp. 279-291.

[27] Wang, M., Yapa Bandara, K. and Pahl, C (2009) Integrated Constraint Violation Handling for Dynamic Service Composition. In IEEE International Conference on Services Computing SCC 2009. pp. 168-175.