Ontology-based Composition and Matching for Dynamic Service Coordination

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

Lero - The Irish Software Engineering Research Centre
School of Computing, Dublin City University, Dublin, Ireland
`[cpahl|vgacitua|mwang|kyapa]@computing.dcu.ie`

Abstract. Service engineering needs to address integration problems allowing services to collaborate and coordinate. The need to address dynamic automated changes - caused by on-demand environments and changing requirements - 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. 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.

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 and service outsourcing [2] pose challenges in terms of flexibility of composition and also scalability.

We introduce an ontology-based solution for service collaboration, focussing on its description logic foundations, that makes a step from static service architectures (based on Web service orchestration) to dynamic coordination [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 next section discusses the context of service collaboration. In Section 3, we address the description of requests and services in the coordination space in terms of ontology-based specification and composition techniques. In Section 4, the matching-based coordination is defined. Section 5 discusses evaluation aspects. We discuss related work in Section 6 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, which requires techniques for the identification of needs and behaviours and the association and customisation of provided services to requested needs [13, 14].

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 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 resultion 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 (help file not OK), 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. Automatically identifying the ongoing process pattern allows a more targeted processing of the initial goal.

The current approach to service composition is to develop service processes that are orchestrations of individual services. 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. Firstly, the inflexible nature: common to both is the static nature of these assemblies, requiring

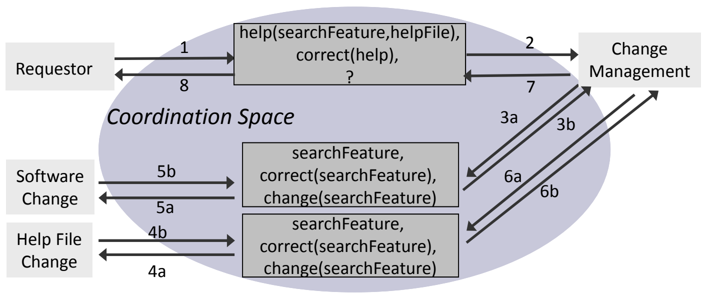

Fig. 1. Coordinated Process of Requests.

them to be pre-defined and 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. Increasing flexibility of composition by allowing partners to dynamically join or leave the community is not possible using the classical approach. The dynamic requesting and providing of services (by asking for activities to be executed on objects to achieve a goal) avoids complex, pre-defined process definitions, thus making the coordination more scalable through self-organisation.

The solution to address flexibility and scalability of collaboration is a coordination space, which acts as a passive infrastructure to allow communities of users and providers to collaborate through matching of requests and provided services. 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 (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.

3 Coordination Request Specification and Composition

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: 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, thus we retain the compositional principle of Web services.

Cloud computing as an information processing and management infrastructure 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 specification and composition of requests.

3.1 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 semantic, goal-oriented specification of objects and processes [17]. We introduce the core of the description 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 activies (object state changes in processes).
- *Individuals* are named objects.

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 *semantics of concept descriptions* as

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

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 4, is a relationship defined by subset inclusions for concepts and roles.

3.2 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, 20]. 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 [21][1].

- Objects types are expected to be 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 [21] 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

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

[1] Transitional roles, which represent state changes and which link objects and their processing from state to state to processes, require a tailored semantics [21].

here requesting the object to be changed such that the type of the component 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.

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

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*.

3.3 Ontology-based Object, Goal and Process Composition

Composition Principles. The core format for request specifications has been defined, which is built on the service process ontology from [21]. We add now support for the compositionality of the request elements, which extends approaches such as [17, 22, 23]. This faciliates 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.

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 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 objects here.
- *Sequential (and behavioural) composition.* Dynamic elements 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.

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 $maintenance \rhd [deploy, analyse, redevelop]$ 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.

[2] Goals are logical expressions, i.e. structural/behavioural composition is not applicable.

– 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).

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 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 4. 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 in Section 3.3.

We extend the semantics 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.

4 Coordination Principles

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:

– deposit(object!, goal!, process!), where the parameters are defined as above in Section 3, 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

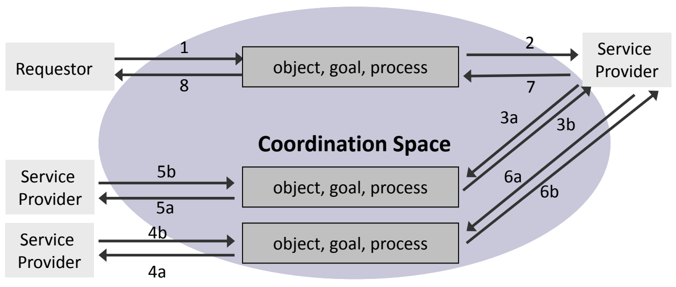

Fig. 2. Abstract Coordinated Process of Requests.

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 equality.

– 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 ontological framework. Subsumption is a relationship defined by subset inclusions for concepts and roles:

– *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 $R_1^I \subseteq R_2^I$.

Subsumption is not implication. Structural subsumption (subclass) is weaker than logical subsumption (implication), see [18]. $C_1 \sqcap C_2 \sqsubseteq C_1$ or $C_2 \rightarrow C_1$ implies $C_2 \sqsubseteq C_1$. We use subsumption to reason about matching of two request descriptions based on transitional roles.

In Fig. 2, 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. The schematic example 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.

While description and reasoning capabilitiues 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.

5 Evaluation

Our key concern here was the definition of an ontology-based language for dynamic request coordination. We have already discussed the theoretical limitations of the language in terms of e.g. decidability earlier on. The sample illustrate the need for the novel components of our approach - the process aspect and the request composition mechanisms, which allow complex taks to be broken up and managed by a specific process. Now, 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 represent dynamic requests as constraints, which need to be validated at runtime. Our environment facilitates constraints generation. We express constraints as CLiX (Constraint Language in XML, `http://clixml.sourceforge.net/`) rules and use the Xlinkit validator engine (`http://www.messageautomation.com/`) to validate them dynamically. This architecture demonstrates the feasibility of ontology-based processing at runtime.

Full behavioural composition is currently not supported. This technology platform has already been used for dynamic service compositions [24], but our approach differs in that it refers to OWL-DL-based ontology request specifications. It uses an analogy between description logics and predicate logics to rewrite the ontology specifications as first-order predicates that are checked by Xlinkit. Our results to-date show an acceptable performance overhead of around normally not more than 10 % for coordination activities (dynamic matching) compared a traditional hardwired WS-BPEL composition of service. The required flexibility gain is achieved (in comparison to a WS-BPEL solution) by enabling dynamic composition.

Another issue is scalability. However, this remains to be investigated further using on-demand systems. We currently have automated process graph generation techniques in place. These shall be used to test the coordination space, firstly, with more complex processes and, secondly, with more processes in parallel. A central success criterion is here the ability to deal with (i.e. match) requests in adequate time (assuming suitable providers).

6 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 [25], 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, 25] by including a process notion as a central component of request tuples [19], 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 provides composition and matching support. Service ontologies are ontologies to describe Web services, aiming to support their semantics-based discovery in Web service registries. WSMO is not an ontology, as OWL-S is, but rather a framework in which ontologies can be created. The Web Service Process Ontology WSPO [21, 26, 27] is also a service ontology, but its focus is the support of description and reasoning about service composition and service-based architectural configuration. An important development is the Semantic Web Services Framework (SWSF), consisting of a language and an underlying ontology [28], which takes OWL-S work [29] further and is also linked to convergence efforts in relation to WSMO. The FLOWS ontology in SWSF comprise process modelling and it equally suited to support semantic modelling within the MDA context. We combine here a process-centric ontology with composition and deploy this in a dynamic composition environment.

7 Conclusions

Manually designed service architectures support software systems in classical sectors such as finance and telecommunications. However, their structural inflexibility makes changes and evolution 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. Scalability can 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. 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.

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 [30]. 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.

Acknowledgment

This research work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie) and grant 07/RFP/CMSF429 CASCAR.

References

1. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web Services - Concepts, Architectures and Applications. Springer-Verlag. 2004.
2. B. Hayes. Cloud computing. Communications of the ACM 51(7):9-11. 2008.
3. J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. Intl. Workshop on Semantic Web Services and Web Process Composition 2004. Springer LNCS 3387, 43-54, 2005.
4. D. Gelernter. Generative Communication in Linda. ACM Transactions on Programming Languages and Systems 7(1):80-112. 1985.
5. E.-E. Doberkat, W. Hasselbring and C. Pahl. Investigating Strategies for Cooperative Planning of Independent Agents through Prototype Evaluation. In P. Ciancarini, editor, Proc. First International Conference on Coordination Models and Languages, Cesena, Italy. Springer-Verlag, LNCS Series 1061, 1996.

6. E.-E. Doberkat, W. Franke, U. Gutenbeil, W. Hasselbring, U. Lammers and C. Pahl. PROSET - Prototyping with Sets, Language Definition. Software-Engineering Memo 15, Universitt GH Essen, 1992.
7. B. Johanson and A. Fox. Extending Tuplespaces for Coordination in Interactive Workspaces. Journal of Systems and Software 69(3), 243-266. 2004.
8. Z. Li and M. Parashar. 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, 104-112. 2005.
9. D. Balzarotti, P. Costa, G.P. Picco. The LighTS tuple space framework and its customization for context-aware applications. Web Intelligence and Agent Systems 5(2): 215-231. 2007
10. L. Nixon, O. Antonechko and R. Tolksdorf. Towards Semantic tuplespace computing: the Semantic web spaces system. In Proceedings of the 2007 ACM Symposium on Applied Computing SAC '07. ACM, 360-365. 2007.
11. NIST. Process Specification Language (PSL) Ontology - Current Theories and Extensions. http://www.mel.nist.gov/psl/ontology.html. National Institute of Standards and Technology, USA. 2007.
12. C. Pahl. Dynamic Adaptive Service Architecture - Towards Coordinated Service Composition. European Conference on Software Architecture ECSA'2010. Springer, LNCS, 2010.
13. C. Pahl, Y. Zhu. A Semantical Framework for the Orchestration and Choreography of Web Services. Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005). Electronic Notes in Theoretical Computer Science 151(2):3-18. 2006.
14. V. Gacitua-Decar and C. Pahl. Automatic Business Process Pattern Matching for Enterprise Services Design. 4th International Workshop on Service- and Process-Oriented Software Engineering (SOPOSE-09). IEEE Press. 2009.
15. A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. Proc. ICSOC06. Pages 27-39. Springer LNCS 4294. 2006.
16. B. Andersson, I. Bider, P. Johannesson, and E. Perjons. Towards a formal definition of goal-oriented business process patterns. BPM Journal 11:650-662. 2005.
17. R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.
18. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.
19. C. Pahl. A Pi-Calculus based Framework for the Composition and Replacement of Components. Proc. Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA'2001 - Workshop on Specification and Verification of Component-Based Systems. Tampa Bay, Florida, USA. ACM Press. 2001.
20. C. Pahl. A Formal Composition and Interaction Model for a Web Component Platform. ICALP'2002 Workshop on Formal Methods and Component Interaction. Malaga, Spain. Elsevier Electronic Notes on Computer Science ENTCS, Vol. 66, No. 4. July 2002.
21. C. Pahl. An Ontology for Software Component Description and Matching. International Journal on Software Tools for Technology Transfer - Special Edition on Foundations of Software Engineering. 9(2): 169-178. 2007.
22. A. Arroyo and M.-A. Sicilia. SOPHIE: Use case and evaluation. Inf. Softw. Technol. Journal 50(12):1266–1280. 2008.
23. C. Pahl, S. Giesecke and W. Hasselbring. Ontology-based Modelling of Architectural Styles. Information and Software Technology 12(1):1739-1749. 2009.

24. A. Dingwall-Smith and A. Finkelstein. Checking Complex Compositions of Web Services Against Policy Constraints. In 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems. 2007.
25. W.T. Tsai, B. Xiao, Y. Chen and R.A. Paul. 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. 175-180. 2006.
26. C. Pahl. A Conceptual Architecture for Semantic Web Services Development and Deployment. International Journal of Web and Grid Services 1(3/4):287-304. 2005.
27. C. Pahl. Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture. European Conference on Model-Driven Architecture - Foundations and Applications ECMDA'2005. LNCS 3748. Pages 88-102, 2005.
28. Semantic Web Services Language (SWSL) Committee. *Semantic Web Services Framework (SWSF).* http://www.daml.org/services/swsf/1.0/, 2006.
29. 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.
30. M. Wang, K. Yapa Bandara and C. Pahl. Integrated Constraint Violation Handling for Dynamic Service Composition. IEEE International Conference on Services Computing SCC 2009. IEEE. 2009.