Semantic Web Service Composition by Consistency-based Model Refinement

Rajesh Thiagarajan Markus Stumptner Wolfgang Mayer
Advanced Computing Research Centre
University of South Australia, Adelaide
{cisrkt,mst,mayer}@cs.unisa.edu.au

Abstract

Modelling complex processes in the Service Oriented Architecture paradigm typically requires the composition of a number of simpler services to achieve a desired goal. We present a semantic service composition approach that is based on the use of UML activity diagrams as abstract specification language and propose a service selection process that combines both conceptual and instance-level analysis to locate suitable services. Model-driven principles allow to adhere to design requirements that would otherwise be difficult to incorporate in formal composition frameworks. Explicit modelling of behaviour in the presence of failures permit to create workflows that are robust in case of service execution errors observed at run-time. This approach may allow to synthesise concrete service orchestrations that are superior to similar approaches purely based on type-based or conceptual matchmaking.

1. Introduction

Pre-planning and manual combination of web services into complex structures is a significant bottleneck on the way to establish dynamic interactions and interoperability, an ability which is one of the main perceived advantages of the service-oriented approach over other paradigms. The goal of providing automated support for web service discovery and composition has led to the development of advanced service description approaches that employ languages which permit defining not just inputs and outputs but also parts of the semantics of the service.

Model Driven Architecture (MDA) offers an abstract framework which can be used to define, in a platform neutral fashion, the various aspects of a system design. The Object Management Group (OMG) recognises UML as a comprehensive system design paradigm. Using MDA-based approaches to assist human comprehension while authoring service descriptions or composing web services is particularly useful since it abstracts the (generally) verbose semantic web formalisms [6, 14]. The effectiveness of using activity diagrams to specify control flow and using a service matchmaker to generate concrete workflows has been demonstrated in [6], where suitable services are located by using input and output (IO) types.

While type-based matching allows to eliminate some services from consideration, in general, a large number of inappropriate services may remain. For example, a request for a book seller service, which takes in a book ID (integer) and a return availability information (string) will be indistinguishable from all services that have the same set of IO parameters. This problem has led to the development of semantic-based matching proposals, where service requests and aspects of a service's functionality are represented formally and suitable services are identified by logical inferences [8, 12, 9].

In this paper, we propose a MDA-based composition approach where abstract workflows describing users' business processes and abstract requirements in an implementation-independent fashion are refined into concrete workflows that are suitable to automatically synthesise and monitor service execution. In contrast to previous work, our matching process utilises both conceptual and instance-level information about services to assess whether a service satisfies given requirements. We demonstrate that our framework complements existing Description Logics (DL) based service selection and allows to synthesise workflows that supersede those derived by purely conceptual matchmaking.

The paper is organised as follows: In Section 2, we present an introduction to MDA-based techniques for the semantic web and motivate the need for a semantic service composition process. Section 3 introduces the notion of consistency-based service discovery, followed by a description of our constraint-based matching framework in Section 4. Section 5 illustrates the approach on an example. In Section 6, related works are discussed and our contributions and conclusions are summarised in Section 7.

2. MDA and The Semantic Web

Model-driven principles have been widely advocated in the software development context, as incremental modelling of complex systems potentially allows to carry out the modelling process in a systematic way and reduce the complexity of the modelling tasks. Abstractions of the various physical system artifacts facilitates a platform independent design process.

We propose to extend MDA-based techniques to the development of service oriented systems, since abstractions of platform and service specific implementation details underlying the semantic web may allow to build complex services more effectively. Furthermore, formalisms already well-known in the software engineering community may allow to leverage already established languages and notations and to abstract from specific formalisms in the Web Service domain. Overall, the model-driven approach may allow to incrementally refine specifications of service orchestrations from the business level down to concrete implementations. Compared to other modelling approaches, such as Petri-nets and BPEL, the use of widely accepted modelling languages may allow a larger community to be involved in the design processes, as well as reducing the effort required to build adequate service descriptions. Furthermore, since UML is likely to be used within most application development processes, our extensions may utilise tools and frameworks that are already deployed. In this paper, UML activity diagrams are employed to model abstract workflows and OCL is used to represent the various requirements of the abstract workflow.

In the proposed framework, the composition process begins with the design of an abstract workflow (Figure 1). An abstract workflow consists of a specification of the control flow between abstract services. An abstract service is a placeholder for one or more concrete services, where requirements and preferences imposed are described as constraints on an implementation-independent level. Our service refinement approach guarantees that the services selected in the concrete workflow synthesised from its abstract counterpart adhere to all constraints given on the abstract level.

For example *booking a hotel room* could be a functional requirement of an abstract service. In addition, the workflow can then be augmented with global (non-functional) requirements and preferences, such as constraints limiting the total costs of a transaction, or preferences for services available from a particular vendor or in a certain geographical region.

Since our workflow specification is based on UML Activity Diagrams, the Object Constraint Language (OCL) is well-suited for expressing abstract requirements and constraints. OCL was originally introduced as a language to represent semantics that are not easily captured through the various diagram notations in UML. OCL has been advocated as a promising alternative to the traditional semantic web formalisms for representing declarative service semantics and service functionality [11].

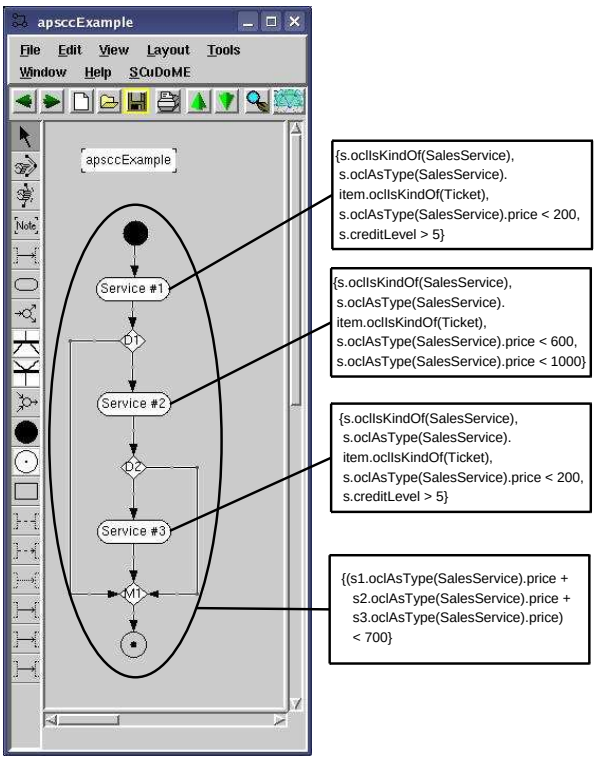


Figure 1: Abstract Workflow Specification

We present a MDA-based composition process that begins with an abstract model of a workflow that specifies the control flow between abstract activities, functional and other requirements of abstract services, and additional global requirements and preferences that may influence the refinement process. This abstract workflow specification is subsequently refined into a concrete implementation, where each abstract service is replaced with a concrete service satisfying the constraints on the abstract level. Throughout the refinement step, both conceptual as well as instance-level properties of services are taken into account to ensure that the best matching service is selected for execution. As a result, a concrete instance of the abstract workflow is obtained that can be used to synthesise a program that executes and monitors the service composition. In case the requirements posed on the abstract level cannot be satisfied by a concrete implementation, if, for example, no combination of services is able to satisfy the given specification, this is indicated to the user who is prompted to relax the given specification.

Formally, the abstract workflow model and its constraints are established by extensions to the UML metamodelling framework. Extension mechanisms in UML, such as profiles, provide platform to extend the UML meta-

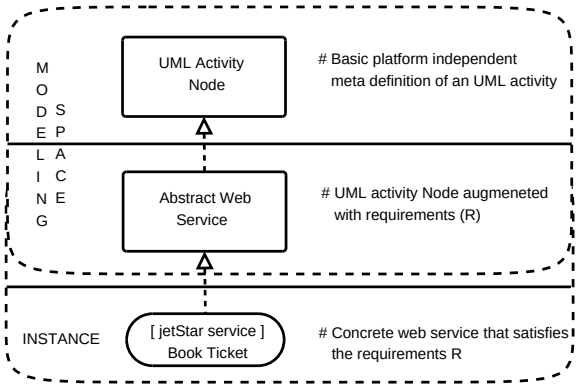


Figure 2: Meta Model Extension

model to include application specific modelling elements. Our proposed extension of the UML meta-model describing activity diagrams is shown in Figure 2. A meta-layer between UML activity diagram meta-model and its instance layer is introduced, where the elements specific to our framework are anchored. *Abstract services* are obtained by specialisation from the generic *Activity node* element in the meta model, such that each node may be annotated with additional requirements.

Requirements in general represent the functionality or the capabilities that are desired for a particular web service. Individual requirements may represent particular services requested, property value restrictions, QoS requirements, or user preferences. All abstract service requirements are modelled in OCL. This allows to leverage existing UML modelling tools for modelling, versioning and to carry out certain model transformations.

3. Consistency-based Service Discovery

Given a formal specification of the requirements of a workflow or individual services, suitable concrete services must be located that are able to fulfil the requested functionality. While early approaches were mainly relying on user-specified informal keywords [4] and types of input and output parameters [9] of a service, these approaches become less effective if a diverse set of services stemming from different sources is to be considered. In particular, syntactic and type-based matching approaches are vulnerable if services use similar type signatures but provide entirely different services. In such scenarios, semantics based annotation and matching frameworks are required to obtain suitable results. Foremost, a number of matchmaking algorithms based on conceptual inferences formalised in Description Logics (DL) have been proposed.

While considerable advances have been achieved using purely conceptual inferences, the abstraction inherent in service profiles and request queries makes it difficult to ascertain whether a matching service will indeed be able to satisfy a given requirement. Furthermore, the result of the matching process heavily depends on the level of abstraction that is used to write profiles and queries. Different notions of *service matches* have been proposed [**?**] to partially overcome some of the difficulties, but remain unsatisfactory to a certain degree due to incompleteness and abstraction in service profiles.

For example, the fact that a service advertises the capability to sell books does not necessarily imply that it will be able to supply *all* book titles. Similarly, the same profile may or may not match a request for the more specific concept *novel*, depending on the matching algorithm being used.

To overcome these limitations, we propose a staged matching process, where conceptual matching is performed to identify a set of potentially matching concrete services that are subsequently queried at the instance level to ascertain if the given service is indeed able to meet the desired requirement. Note that our approach may be employed online by means of directly querying a concrete service via its publicly available API, as well as off-line, where a database stating the precise capabilities of a service is used to confirm or refute a particular query. Note that while the profile at the conceptual level may contain abstract concepts and omit many details of a service's capabilities, it is assumed that the concrete instance-level profile contains a specification that is precise enough to definitely answer a query. This allows to apply the closed world assumption, such that if a particular service instance cannot be shown to definitely match a given requirement, a mismatch is assumed and another concrete service must be found.

The service discovery process proposed in this work is based on the notion of *logical consistency* between a service profile P and a service requirement R: P is considered a potential match for R iff $P \wedge R$ is satisfiable. It is assumed that the service profiles and requirements have been translated from OCL into an equivalent representation in FOL [3].

In contrast to the stronger equivalence, subsumption or plugin matches [8], consistency as the basis for matching provides the advantage that a service profile P is considered a potential match for a request R if P *may* satisfy R. A disadvantage of this criterion is that more matches may be returned. To counter this problem, the instance-level matching phase eliminates services which cannot provide a requested service. While some aspects of this type of reasoning can be achieved by translating aspects of instance-level service profiles to the conceptual level, the large number of concepts and axioms required is beyond the capabilities of existing DL reasoners.

For example, consider a service profile that advertises

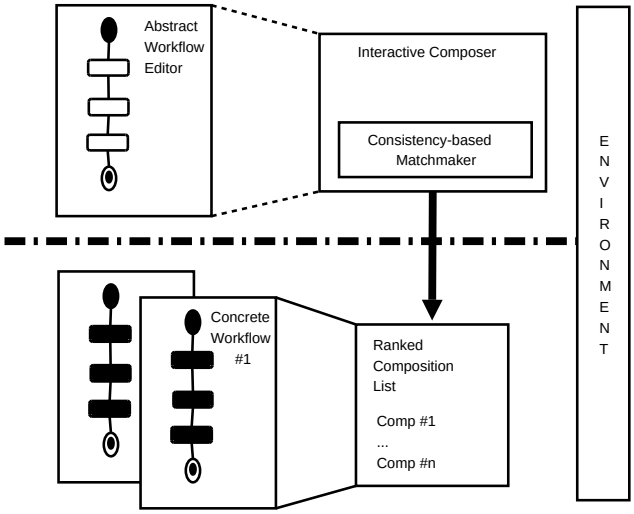


Figure 3: Composition Process

that a particular service sells airline tickets between Australian cities. While there may be a number of services (e.g. Travel agencies) that can indeed provide tickets for any combination of cities, some services may provide this service only between particular destinations. Yet, at a conceptual level, it would not be unreasonable to advertise this service as providing tickets between Australian cities. Therefore, such services would match a given query even for destinations where the service will not be adequate. However, this fact will not be provable on the conceptual level. In this case, the service instance must be queried to ensure the destinations in question are actually available.

The overall architecture of our model-driven development framework is shown in Figure 3. Given an abstract workflow and requirements developed in a standard UML development environment, a consistency-based refinement process is carried out, leading to a set of concrete service orchestrations. Since we aim at a platform-independent architecture, the services that are available for instantiation by the consistency-based matchmaker are represented as an abstract *Environment*, which provides service profiles to drive the discovery process. Here, existing technologies such as OWL-S, BPEL, and UDDI may be employed as underlying implementation platform. As a result, a set of concrete workflows with instantiated services is obtained and presented to the user.

While consistency-based matchmaking is well-suited to consider "firm" constraints, "soft" user preferences that may be violated to obtain a solution require complementary approaches. For simplicity, we abstain from defining our own ranking framework to order different solutions and utilise existing work on recommender systems [1] to provide adequate ranking functions, taking into account user preferences specified in the abstract workflow.

4. Constraint-based Service Discovery

Constraint-based systems, which have been successfully applied to a number of large-scale industrial problems, are also useful in the web service composition context. Previous works, in particular [2, 7], have examined the feasibility of using constraint-based systems to address the web service composition problem. While [2] provided a constraint-based workflow design based on input and output compatibilities, [7] provided an optimised approach taking into account simple attributes of services.

The need for a matchmaking process that filters web service candidates based on the functionality a web service offers has already been established. However, the previous constraint-based proposals do not consider semantics such as functionality of a web service while performing candidate selection. In this section, we present our MDA-based service composition approach, formalised in terms of a constraint-based system to generate concrete workflows for any given abstract workflow.

In difference to previous work, where similar services were implicitly assumed to require the same set of IO parameters, our generative formalism allows to tolerate differences in IO signatures between services providing similar functionality. This allows to adapt the control and data flows devised from an abstract workflow, leading to more flexible refinements and the possibility of compensating for failures detected at run-time by reconfiguring the remaining orchestration.

The profile of a service is an advertisement of the capabilities and properties that a service claims to offer. A profile is formalised using a set of constraints over attributes of the service. The constraints in a profile P should be satisfied by all the concrete service instances sharing P.

Definition 1 (Service Constraint) *A service constraint p is defined as a pair $p = \langle A, C \rangle$ where A is a set of attribute names corresponding to instance-level properties of a service and C is a set of constraints specifying the conceptual properties of a service. A service constraint $q = \langle A', C' \rangle$* specialises *$p$ iff $A \subseteq A'$ and $C \subset C'$.*

In this paper, it is assumed that the constraint language used to express constraints in $p = \langle A, C \rangle$ is OCL restricted to using only attribute names in A. The semantics of a set of constraints $C = \{c_1, \ldots, c_n\}$ is given by the conjunction $c_1 \wedge \ldots \wedge c_n$. As mentioned previously, this subset of OCL can be translated to FOL to allow efficient inferencing [3].

The notion of service constraint can be used to define *service profiles* as well as *service requests*.

Example 1 *A sales service profile p to advertise sales of some kind of $Book$ at a price of at-most \$30 is modelled as* $A = \{item, price\}$
$C = \{item.oclIsTypeOf(Book), price \leq 30\}$. *The same*

service constraint could be interpreted as request for a service selling books worth not more than $30.

All the concrete services that use a profile p as their advertisement should satisfy C. Intuitively all the services that advertise p are services that sell some kind of $Book$ worth at-most $30.

The difference between concept-level and instance-level profiles is given by the constraints in C. While complex predicates and relations are permitted on the conceptual level, instance level profiles are restricted to equality constraints on property values.

While profiles at the instance level describe precise service attributes that cannot be specialised further, conceptual profiles may represent abstract specifications that allow more than one specialisation to obtain a concrete service. The hierarchy of profile descriptions forms a lattice structure, with $\top$ denoting the completely unconstrained profile and $\bot$ representing an infeasible profile.

To discover advertised services, we assume that a registry (e.g., UDDI or semantic registry) exists that publicises service profiles. To abstract from implementation details, we define an abstract environment:

Definition 2 (Environment) *An environment E is defined as a tuple $\langle S, E_C, E_I \rangle$ where, S is a set of concrete services, E_C is the set of conceptual service profiles for services in S, and E_I denotes the set of instance level profiles over S. Let $users(p) \subseteq S$, $p \in E_I$, denote the set of services that conform to instance-level profile p.*

All concrete services in an environment are associated with a single profile in E_C and E_I, respectively. Despite the similarity of instance level profiles and concrete services, a profile may be associated with more than one service in case equivalent (replicated) services exist that provide the same functionality. An example environment for our business travel workflow is presented in Figure 4.

Consistency-based candidate selection requires to retrieve a set of concrete services such that their conceptual or instance level profiles satisfy a given service request R. Formally, we define an entailment relation $\models$ to ascertain whether a profile in the environment matches a given service request:

Definition 3 (Satisfying Profile) *A service profile $p = \langle A, C \rangle \in E_C \cup E_I$ satisfies a service request $r = \langle A', C' \rangle$ iff*

$$C \wedge C' \not\models \bot$$

That is, a profile p potentially satisfies a request r if p or one of its feasible specialisations satisfy r. If $p \in E_I$, then it can be established precisely whether p satisfies r, assuming that the constraints in r are limited to attributes in p; otherwise, a common model q of both p and r may exist, but $q \notin E_I \cup E_C$.

Definition 4 (Abstract Workflow) *An abstract workflow is defined as a tuple $\langle N, \epsilon, \Omega, T \rangle$ where $N = A \cup V$ denotes a set of nodes, A denotes the set of abstract services, V represents the set of control nodes and $T \subseteq N \times N \times L$ denotes labelled transitions between elements in N, where L denotes a set of OCL constraints representing the transition condition. The set $\Omega \subseteq V$ represents the set of final states with no outgoing transitions, whereas $\epsilon \in V$ denotes the unique initial state. The control nodes in V represent choice and fork nodes in the workflow's control structure.*

Each abstract service is associated with an (abstract) service request that restricts the services that may be used to specialise the workflow. In this work, only acyclic workflows are considered.

For simplicity, we assume the initial and final states are represented as auxiliary services. While the initial state is associated with a profile that supplies the input parameters of the workflow, each final state represents a possible termination point of the workflow. Since multiple exit points may be defined, different requirements on normal completion, that is global requirements on the entire workflow, as well as failure conditions may be specified. This allows to restrict the refinement process to services that conform to desired behaviour in case of failures.

Starting at the requirements associated with terminal states Ω in a workflow, the requirements stated for individual abstract services are inferred by propagating requirements backwards along transitions. For each transition t that is traversed, the requirements are amended with the transition condition associated with t. For choice nodes, the disjunction of the requirements propagated from the outgoing transitions are propagated along the incoming transition. Combined with local requirements and preferences associated with an abstract service, the resulting abstract service request is used to select suitable specialisations for service nodes in the workflow. The profile associated with the input node of the workflow must meet the requirements propagated to the node. For nodes representing abstract services, aspects of the requirements that are satisfied by a matching service profile are removed from the requirements propagated further. If no match can be found, the most comprehensive requirement $\top$ is propagated. If the input node cannot meet its requirements, no solution exists and a different refinement alternative is considered. If no consistent refinement is found, the user is notified and given the opportunity to relax the workflow specification. Formally, the propagation algorithm can be implemented in the open constraint satisfaction framework proposed in [5] or the generative CSP framework used in large-scale configuration systems [2, 10].

As outlined in previous sections, the service selection process is performed iteratively: first, the requirement constraints r are used to identify potential matches with concept-level profiles in E_C in a given environment. Match-

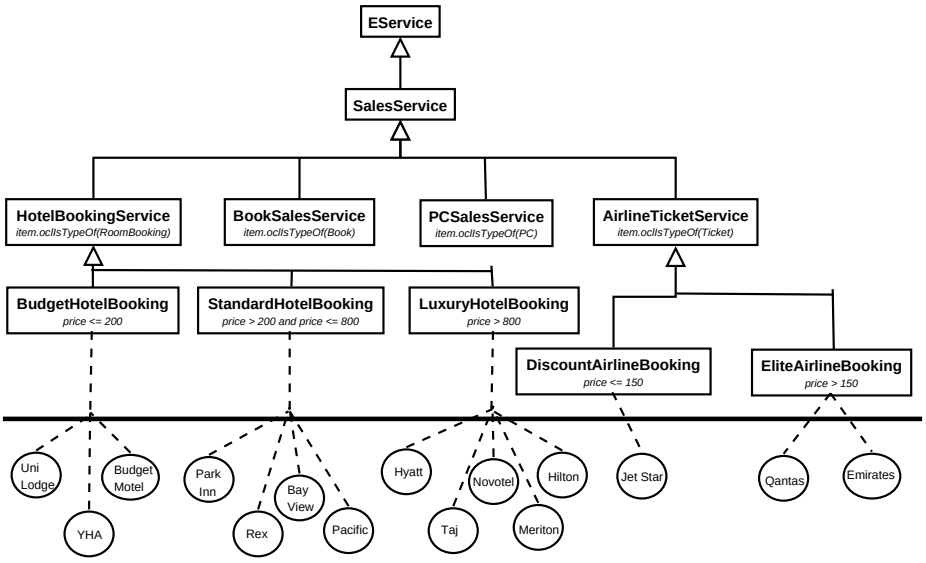


Figure 4: Conceptual Service Model and Instance Profiles

ing profiles are specialised to obtain a set of instance level profiles, which are subsequently matched against r to isolate those service profiles $M = \{p_1, \ldots, p_n\}$ which indeed offer the desired service. From instance-level profiles in M, the set of matching services $\bigcup_{i=1}^{n} users(p_i)$ is obtained and concrete candidate workflows are instantiated. For efficiency, only the most preferred scenarios are considered, based on the users' preferences. As a result, a workflow comprised only of concrete services is produced that may directly be executed.

The composition approach has been partially implemented in Cincom Visualworks/Smalltalk. The abstract workflow modeller and composition viewer are both implemented as extensions to Honeywell's DOmain Modelling Environment (DOME)[1] which facilitates definition of arbitrary diagram notations on the basis of a meta class notation.

5. Example

We use the following business travel booking scenario to demonstrate our composition approach. The requirements for the business travel workflow are as follows.

Requirement 1 *The following service requests r_i, global requirements G, and predefined control flow CF are the requirements of our travel workflow.*

r_1**:** *Make an airline booking starting from Adelaide on the 23/11/07 to Melbourne and do not spend more than $200. The reliability of the service provider should at least be 5.*

r_2**:** *Make an hotel reservation for stay at Melbourne. Checking in on 23/11/07, checking out on 24/11/07 and spend anywhere between $600 and $1000.*

r_3**:** *Make an airline booking starting from Melbourne on the 24/11/2007 to Adelaide and do not spend more than $200. The reliability of the service provider should at least be 5.*

G**:** *The overall costs should not exceed $700.*

CF**:** *If either s_1 or s_2 is not successful then do not perform s_3 and terminate the workflow.*

The travel workflow requirements from above can be modelled into an abstract workflow as shown in Figure 1.

We assume the environment shown in Figure 4 and assume its state from Tables 1 and 2. The composition process proceeds as follows:

1. The requirements given above are modelled into an abstract workflow. The service requests $r_1, \ldots, r_3$ are modelled as abstract services $s_1, \ldots, s_3$ with individual requirements in OCL. The global requirement (overall costs) is modelled as a constraint at the exit node of the workflow. The final abstract workflow specification is shown in Figure 1.

2. Given the abstract workflow and the environment, the composition process begins by propagating the global constraint G over to 'Service #3' (s_3).

[1] http://www.cis.unisa.edu.au/~cisgg/wiki/dome/index.html

Service ID	Service	PM	PS
1	Uni Lodge	101	101
2	YHA	90	90
3	Budget Motel	170	170
4	Park Inn	201	201
5	Rex	301	301
6	Bay View	501	501
7	Pacific	601	601
8	Hyatt	899	899
9	Taj	999	999
10	Novotel	849	849
11	Meriton	949	1099
12	Hilton	1199	1120

PM – Price at Melbourne, PS – Price at Sydney

Table 1: Hotel Reservation Services State: E_{hotel}

Service ID	Instance	A-M	M-A	S-A	CL
13	Jet Star	115	51	201	6
14	Qantas	145	101	251	8
15	Emirates	175	121	301	9

A-M – Adelaide to Melbourne, M-A – Melbourne to Adelaide, S-A – Sydney to Adelaide, CL – Credit Level

Table 2: Airline Services State: $E_{airline}$

Subsequently the two step matchmaking procedure is invoked with requirements from s_3. In the first step, relevant conceptual-levels from the environment are shortlisted and the resultant set is {DiscountAirlineBooking,EliteAirlineBooking}. In the next step instance-level profiles that are specialisations of previously shortlisted concept-level profiles are further queried to determine a set of instance-level profiles that indeed satisfy the requirements. The concrete services that use the determined instance-level profiles are returned as matches for the given request.

3. The previous step is carried out on all the abstract services by navigating backwards over transitions while propagating the constraints.

4. For this example there are no compositions such that G is satisfied. The user is notified that the requirements are over-constrained (Figure 5a).

5. Assume that G is revised to *The overall costs should not exceed $800* G'.

6. A rerun of the composition process with the revised constraints results in a set of concrete workflows ordered according to the total cost (see Figure 5b).

7. One of the concrete workflows can then be selected for viewing (see Figure 5c).

6. Related Work

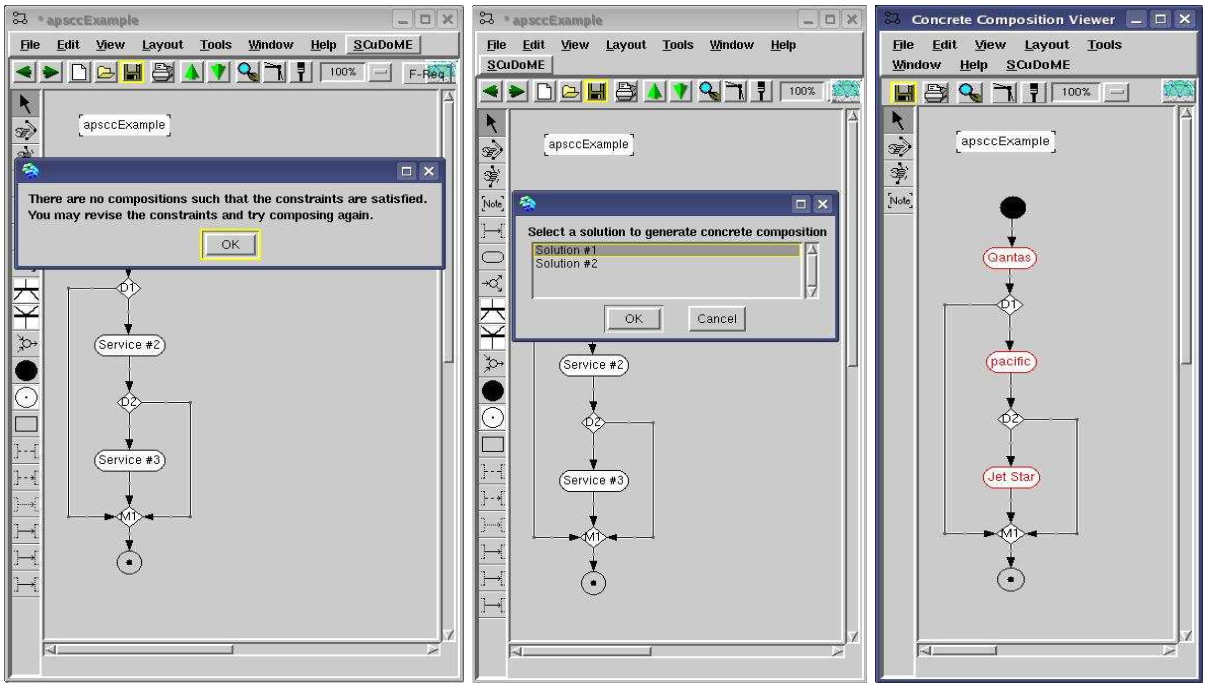
A model-based approach for the development of syntactic service descriptions was proposed in [14]. The primary contribution is an abstract service description approach that is transparent to platform specific details. Representation of complex semantics such as capabilities are not addressed. In the approach presented here, service descriptions are augmented with capability descriptions which are further used within a composition framework while formulating complex workflows.

UML activity diagrams are used as a starting point in [6] to perform web service composition. Matches are determined based on subsumption relationships between IO types. It is likely that matchmaking of this kind could result in unintuitive matching where the returned services do not provide the required functionality. Our work explicitly considers complex semantics of a service within our composition framework.

A number of approaches that use logic reasoners [12, 8] or other formalisms such as object-action semantics [13] have been proposed. However, these approaches are only able to check whether an advertisement *may* be relevant to a request. These frameworks are vulnerable to ambiguity in conceptual modelling and limitations of current inference engines. In contrast, the two step matchmaking scheme proposed here builds on techniques that have been proved to handle large scale problems, while avoiding undesirable aspects of concept-level reasoning.

7. Summary and Future Work

We presented a model-driven service composition approach powered by a semantic candidate selection scheme, where concrete service orchestrations are synthesised from abstract UML diagrams and OCL specifications of the desired functionality. A two-phase semantic matching process is employed, where concept-level reasoning is carried out to isolate suitable candidate services, while instance-level reasoning provides precise filtering to isolate those service that are indeed suitable. The resulting workflows are potentially more precise than those obtained by concept-level reasoning alone. Flexible handling of differences in IO signatures of similar services allows for adaption and refinement of abstract and concrete workflows, allowing to automatically react to failures observed at run-time. Widening the composition process to account for different types of requirements specifications and refinement schemes, as well as extending investigating different approaches to incorporate user preferences are among future research. While considerable progress has been achieved since the service oriented paradigm was first introduced, the current state of the art is far from satisfying the goals of the original vision. In particular, means to ease the service development process to en-



(a) Over-Constrained (b) Ranked Compositions (c) Concrete Workflow

Figure 5: Interactive Composition Process

sure that services are advertised suitable as well as queried properly should be developed to aid further proliferation of concrete implementations. The architecture presented in this paper can be considered a first step towards leveraging existing software engineering technologies to ease this process.

References

[1] E. Al-Masri and Q. H. Mahmoud. Discovering the Best Web Service. In *Proceedings of the 16th International Conference on the World Wide Web*, pages 1257–1258, Banff, Canada, May 2007.

[2] P. Albert, L. Henocque, and M. Kleiner. Configuration Based Workflow Composition. In *Proc. of ICWS 2005*, July 2005.

[3] B. Beckert, U. Keller, and P. H. Schmitt. Translating the Object Constraint Language into first-order predicate logic. In *Proceedings, VERIFY, Workshop at Federated Logic Conferences (FLoC)*, Copenhagen, Denmark, 2002.

[4] T. B. et al. UDDI Version 3.0. Technical report, UDDI.org, July 2002.

[5] B. Faltings and S. Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161(1-2):181–208, 2005.

[6] R. Grønmo and M. C. Jaeger. Model-Driven Semantic Web Service Composition. In *APSEC05*, pages 79–86, Dec. 2005.

[7] A. B. Hassine, S. Matsubara, and T. Ishida. A Constraint-Based Approach to Horizontal Web Service Composition. In *Proceedings of the International Semantic Web Conference (ISWC)*, Athens, GA, USA, Nov. 2006.

[8] L. Li and I. Horrocks. A software framework for matchmaking based on Semantic Web technology. In *Proc. of WWW 2003 Conf.*, pages 331–339, Budapest, May 2003.

[9] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *Proc. ISWC*, pages 333–347, Sardinia, Italy, June 2002.

[10] M. Stumptner, G. Friedrich, and A. Haselböck. Generative constraint-based configuration of large technical systems. *AI EDAM*, 12(4), Dec. 1998.

[11] R. Thiagarajan and M. Stumptner. A native ontology approach for semantic service descriptions. In *Second Australasian Ontology Workshop (AOW)*, volume 72 of *CRPIT*, pages 85–90, Hobart, 2006. ACS.

[12] H. Wang and Z. Li. A Semantic Matchmaking Method of Web Services Based On $\text{SHOIN}^{+}(\text{D})^{*}$. In *Proceedings of the IEEE Asia-Pacific Conference on Services Computing (APSCC)*, pages 26–33, Guangzhou, China, Dec. 2006.

[13] L. Ye and B. Zhang. Discovering Web Services Based in Functional Semantics. In *Proceedings of the IEEE Asia-Pacific Conference on Services Computing (APSCC)*, pages 348–355, Guangzhou, China, Dec. 2006.

[14] B. Yu, C. Zhang, and Y. Zhao. Transform from Models to Service Description Based on MDA. In *Proceedings of the IEEE Asia-Pacific Conference on Services Computing (APSCC)*, pages 605–608, Guangzhou, China, Dec. 2006.