A Conceptual Modeling Approach for Web Service Composition Supporting Service Re-Configuration

Georg Grossmann **Michael Schrefl** **Markus Stumptner**

University of South Australia, Advanced Computing Research Centre, Mawson Lakes, SA 5095, Adelaide, Australia. E-mail: {georg.grossmann, michael.schrefl, mst}@cs.unisa.edu.au.

Abstract

Service composition is a recent field that has seen a flurry of different approaches proposed towards the goal of flexible distributed heterogeneous interoperation of software systems, usually based on the expectation that such systems must be derived from higher level models rather than be coded at low level.

We propose a conceptual modelling approach for the composition of service processes into a task workflow and its dynamic adaption to changes during runtime. The contribution of the paper is twofold. Firstly, our approach improves on existing approaches by the separation of configuration into service configuration and service instance configuration which reduces the reconfiguration cycles in case of changes and secondly, we describe a comprehensive modelling concept that combines existing dynamic composition techniques in a novel way.

1 Introduction

The dynamic interaction of the software systems that support modern business processes is among the most important challenges faced by organizations today. A key issue faced by the developers in such situations has always been that, inevitably, separately developed systems rely on disparate models and no matter how flexible the infrastructure, it requires major effort to establish interoperability via the mutual adaption of coexisting business processes, or the integration of legacy applications, or the incorporation of local "grassroots" solutions in individual branches of enterprises that often have thousands of applications and data repositories spread across the organization. In most cases nowadays, this integration task is addressed by manual analysis and laborious development of specific integration solutions.

This problem, traditionally experienced in the context of interaction and distributed database systems, has become particularly poignant in recent times with the explosive development of new styles of infrastructure that support significantly more flexible and dynamic interactions between applications. The so-called Service Oriented Computing (SOC) paradigm uses the idea of flexibly (possibly dynamically) assembling individual application components (possibly directly accessible via the Web) into complex processes. The underlying technology on which SOC is often assumed to be based is the technology of Web Services, using Web-based communication protocols and XML-based data formats and communication interface definitions, all of which are enjoying increasingly widespread usage. However, unless everyone uses the same data structures and interfaces, a utopian scenario, fully realizing the SOC dream requires overcoming the interoperability problem [25].

Four aspects discussed in related literature in the context of SOC are abstraction, semantic description, service behaviour and dynamic integration [7, 24]. Neŭcaský et al. propose the abstraction from underlying XML-based description and using a single ontology derived from existing descriptions to facilitate the semantic integration [24]. Similarly, the BPMO approach proposes the use of an ontology but considers in addition the service behaviour [7]. Our approach goes beyond that by considering the re-configuration of services if the environment changes during runtime.

The context of this paper is the interoperation of several service processes that are dynamically composed into a *task workflow* [12] to achieve a certain business goal. We use the notion "'task workflow"' because it used in related literature but point out that a task in a task workflow may consists of subtasks modelled as activities and subactivities. For example, a simple task workflow could be a travel booking, comprising a flight and a hotel booking. The relevant service processes would be flight reservations and bookings of different airlines and hotel reservations and bookings of different hotels.

Semantic interoperation is the process of bridging and reconciling heterogeneity for combining a set of service processes, selected based on their capabilities, to meet a goal that cannot be met by using a single service. The composition of the services of the combined service processes needs to be planned and the composed services needs to be enacted. This is described by a workflow of itself, the *task workflow* [12].

Pre-planning and manual combination of service processes 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, employing languages which permit defining not just inputs and outputs but also parts of the semantics of the service. In practice, however, even apparent front-runners can prove to be difficult to apply [30, 4] and insufficient for describing certain complex functionality [9].

A fundamental decision concerns the selection of the underlying interoperability architecture and knowledge representation of service processes and the composite task workflow. The representation must be rich enough to capture the semantics of service pro-

This research was partially supported by the Australian Research Council (ARC) under grant DP0988961.

cess profiles in order to facilitate the detection and bridging of semantic heterogeneities, abstracting from syntactic details of Web service descriptions. The architecture should support different target implementations. An approach that was developed with this goal in mind is the so-called Model Driven Architecture (MDA). It offers a layered architecture that permits modeling of system artifacts in a platform independent fashion, with the goal of automatically mapping them later to a specific implementation platform. MDA representations typically (though not always) use variants or subsets of UML. UML has been used for modeling service workflows, e.g., [12], but again only at the structural level. UML does provide more powerful means of expression, such as the Object Constraint Language OCL, which has been successfully used for knowledge representation [9], including in the SOC area.

The contribution of this paper lies in a top-down design approach for the dynamic composition of Web services. It consists of a comprehensive modelling concept that combines existing dynamic composition techniques and provides a multi-level configuration approach on the service and service instance level that reduces the reconfiguration cycles.

The next section describes a motivating example followed by a discussion on related work. Section 4 explains the composition approach in detail followed by the conclusion.

2 The Business Trip Scenario

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

r_1: Make a round trip airline booking from Adelaide to Vienna on either one of the airlines Deepblue or PurpleRed, with preferred departure date 4th of July and preferred return date 14th of July, and make an accompanying hotel reservation with StarHotels.

r_2: Spend maximum 2,800 AUD for the airline ticket.

r_3: Select a hotel rated as $***$-stars or higher, spend a maximum of 1,900 for accommodation.

r_4: The overall trip costs should not exceed 4,600. If overall costs can be reduced by more than 10% percent, the travel itinerary may deviate from the preferred dates, with earliest departure 2nd of July, latest departure 6th of July, earliest return 12th of July, latest return 18th of July, a minimum stay of 10 days and maximum stay of 14.

3 Related Work in Dynamic Composition of Services

Semantic Web service composition is the process of combining a number of Web services, selected based on their capabilities, to meet a goal that in general cannot be met by using a single service. Web service compositions are specialized business processes or workflows where every activity or operation in the process is carried out by a Web service. Previous works have examined a number of traditional paradigms such as planning [19, 26, 34, 32], ontology-based matchmaking [23, 11, 20] and constraint-based systems [2, 16] to automate various parts of the composition task. Alternatively, some approaches such as [12, 36] have investigated the effectiveness of using MDA-based technologies to accomplish semantic Web tasks. In this section, we discuss some related approaches and techniques in the area of service configuration. Further work can be found in the modelling of adaptive workflows in the health care domain [6, 35], concepts and tools for modelling dynamic workflows and process variants [21, 1, 28] and research in self-healing business processes and Web services [15].

Related Approaches: There are multiple approaches developed by the research community in dynamic Web Services composition but only few address a top-down design. We discuss the four approaches that are more closely related to the approach presented in this paper: (1) Processes with Adaptive Web Services (PAWS) [1] [3], (2) ServiceMosaic [5], (3) the Model Driven Web Service Composition developed by Grønmo et al. [12] and (4) the Service Delivery Life-Cycle for Semantic Service Provisioning [18, pp.14-18].

The Process with Adaptive Web services (PAWS) framework couples adaptation design-time and run-time execution [3]. It presents an approach that deals with service processes and provides several design- and run-time tools such as mediator configurator and mediator engine for coupling modules and supporting message transformation. PAWS differs in achieving flexibility because of providing advanced design tools and separating the design from the run-time part in the composition process. This has the disadvantage that the number of services as well as all possible exceptions must be known prior to execution. Our approach allows to re-configure the design of a service process in case of a run-time failure and to consider new alternative services in the composition.

The ServiceMosaic project targets the development of model-driven adapters for business protocols [5]. In this project Benatallah et al. investigated the automated analysis of compatibility and replaceability of services on the protocol level as basis for the model-driven development of Web service protocol adapters. Adapters are used to mediate between incompatible interfaces of functionality-wise compatible services and adapting Web services in a way so they become compliant with other services. This work relates to one part of our composition approach, the service configuration step described in Section 4.3 in which mediators are developed in order to communicate with the interfaces of service providers.

Grønmo et al. propose a model-driven composition approach for semantic Web services [12]. It considers the discovery and selection of service using an ontology based approach in order to capture the semantics of service functionality. The approach also supports the definition of Quality of Service (QoS) in form of requirements such as maximum costs. However, it does not consider the reconfiguration of the composite process if unforeseen changes of the service environment occur.

Grønmo et al. introduced the concept of an abstract workflow in the composition approach. An abstract workflow is a pre-specified control flow specification between abstract service specifications. UML activity diagrams are used to model the abstract workflow. Abstract service specifications, which are a representation of a task that needs to be realized, are modelled as activities. Given an abstract workflow, the composition process discovers services from a registry that meet the requirements, replaces abstract services with respective concrete services, and finally formulates a concrete workflow where all the abstract service specifications are replaced with a respective concrete service while retaining the control flow from the abstract specification.

[1] http://www.paws.elet.polimi.it/

Kuropka et al. describe an approach for *adaptive service provisioning* called *service delivery cycle* [18, pp.14-18]. The life cycle considers the iterative planning and enactment of services, where the planning sub-cycle includes matchmaking and composition, the binding sub-cycle includes negotiation and contracting, and the enactment sub-cycle includes invocation, monitoring and profiling. This approach overcomes some limitations of static binding of Web services, e.g., it allows the utilisation of new Web services during the enactment and can accommodate changes such as sudden unavailability of services. Our approach goes beyond this by distinguishing between the composition and matchmaking on the schema and instance level and proposing a configuration rather than a planning approach. A configuration on the schema and instance level has the advantage that a composition can be adapted by reconfiguring bound services in a subcycle rather than performing a new replanning and rebinding cycle with all available services every time a change occurs. For example, in the business trip scenario it is possible to identify an airline that fulfills all the criteria on the service description level. However, it might happen during runtime that there is no flight available on a specific day. Instead of replanning the whole trip, an alternative flight on a different day is searched with the already chosen airline.

The problem that many planning approaches face is that the number or types of services involved in the composition process must be known beforehand because they have a finite set of variables. These predictions are usually difficult to make especially for dynamic composition scenarios. Therefore we propose a configuration approach instead of a planning approach which defines the scenario in form of a *Generative Constraint Satisfaction Problem* [22]. The approach is explained in more detail in Section 4.3 and overcomes the limitation previously mentioned.

Service Configuration: AI planning approaches have been proposed as one option for automated Web service composition [17, 18]. In planning, an initial and final state are provided along with constraints on actions that are available to agents. In a forward chaining approach, the agent starts in the initial state and tries to identify the next action which brings the solution closer to the final state, in contrast to backward chaining which starts in the final state and tries to identify actions backwards in a possible control flow leading to the initial state.

An alternative approach is presented by consistency-based configuration that has been widely studied and applied to a number of industrial problems. Web service composition can be modelled as a constraint satisfaction problem as shown in [2, 16, 33]. In comparison to other approaches this work is robust in handling overspecified profiles which usually lead to less preferable matches and in supporting queries for the non-existence of certain properties.

4 Dynamic Composition and Re-adjustable Execution of Service Processes

The interoperation of business processes is generally acknowledged as one of the most challenging problems faced by the more and more interconnected digital economy, still requiring at this point largely manual intervention resulting in huge costs during the development and over the lifetime of software systems. The shift to more flexible and increasingly virtual technologies does not alleviate the problem, rather it runs the risk of not realizing the potential gains inherent in these technologies if the fundamental interoperability problem is not tackled. The approach outlined in this section uses a combination of novel extensions of technologies that have, individually, been proven to be very powerful in tackling difficult real-world problems as demonstrated by the authors' earlier work [14, 31, 10].

We propose to use a Model-Driven Architecture (MDA) approach as input to a configuration reasoning engine to create interoperable business processes. Figure 1 shows an overview of the composition during its life-cycle.

The actual integration of services becomes an integral part of a workflow that is seamlessly embellished with matching-level activities as needed. As a result, the workflow is dynamically built and adapted while using a straightforward notation. Once assembled, a workflow is reusable, so that many bookings can theoretically be made subject to successful execution. This successful execution is adaptive to environment changes (e.g., selected hotels having no rooms free once the user has made a decision). However, there is no explicit exception handling, instead we provide first class workflow activities ("reflective workflow") to enable adaptation at runtime such as re-planning a business trip if a hotel room previously quoted is no longer available at booking.

Most importantly, the modeling approach is intuitive: the process is modelled the way a normal clerk would do it, but the cumbersome matching task is automated. The user is in control through providing parameters and priorities. If a library of task workflow patterns exists for a particular domain, different patterns could be selected according to preferences (e.g., book hotel first, or check multiple airlines etc).

Semantic technologies assume the existence of a machine-interpretable description of information and services. In our case, the "semantic" aspects refer to three specific technologies linked up to provide a joint model of the problem area:

- an underlying object model of the entities and relationships in the problem area, amended with declarative constraints between these entities and a specific set of relationships that permit a meaningful distinction between different types of heterogeneities [14].
- a model of the behavior of a set of processes, expressed in a formally defined diagram language (Petri net-like or UML activity diagram extended with states) [27].
- a declarative approach for reasoning about these processes and the constraints in the object model in a seamless fashion [33, 2, 8].

The composition process consists of three levels, (1) abstract level, (2) candidate selection and (3) execution level. They are based on the abstraction levels proposed by the MDA but differ in the way that they might not be executed in a strict sequence but interleaved. Under certain circumstances explained below, it might be necessary to loop back to a previous level. The approach

- begins on the abstract level with a user modeling a task workflow with
  - a pre-specified control flow between "service usage tasks", each tied to an "abstract service object" (e.g., a flight reservation)
  - functional and other individual requirements of every service usage task (e.g., payment in Euro or in AUD), and

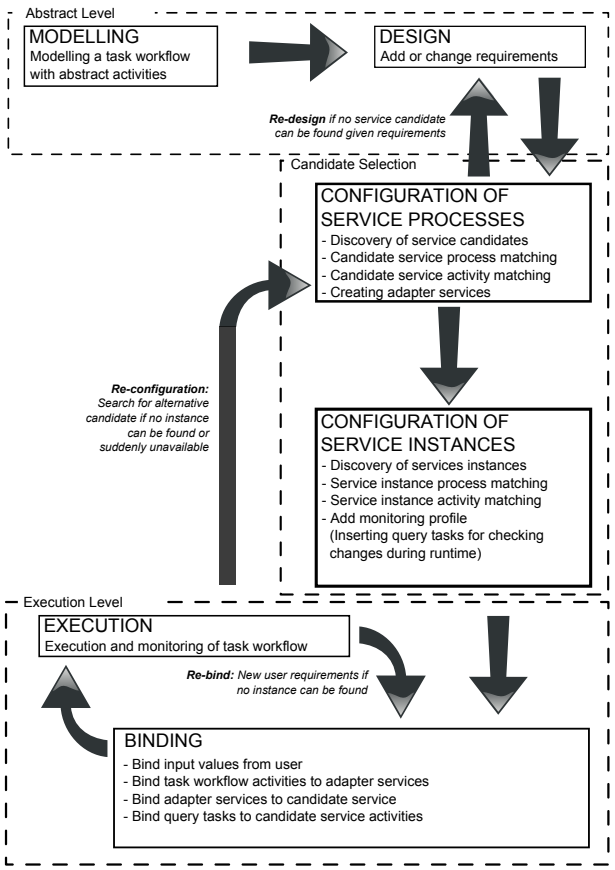

Figure 1: Overview of approach for dynamic composition and reconfiguration during runtime.

  - additional global requirements of the whole workflow (e.g., maximum cost)

- proceeds with a candidate selection which includes the identification and interoperable integration of service processes and their instances into the task workflow, whereby
  - semantic heterogeneities are resolved through inclusion of "bridging tasks" (e.g., an activity converting different currencies), or composing tasks into compound non-decomposable task (e.g., payment and ticketing), or re-ordering tasks (e.g., payment before ticketing), if appropriate and acceptable to the user, and
  - "control tasks" are inserted to query service processes for their current service environment, i.e., their available or running service instances, (e.g., available flights or status of flight bookings), to match - at the instance level - service instances that best meet the specified global requirements to abstract service objects, and to configure or re-configure (upon a changed service environment) a corresponding workflow execution schedule (determining which paths to take in the workflow "upstream" of the currently active task),

- proceeds with the execution, whereby
  - user input values (e.g., flight details) are bound to task workflow and abstract service objects are bound to selected service instances, and
  - "service usage tasks" are subsequently executed.

Some steps in the composition process may be repeated if changes occur:

- *Re-design*: If no service candidate can be identified during the candidate selection because the requirements cannot be fulfilled by available services, the user has to change the requirements on the abstract level (e.g., change order of service usage tasks).
- *Re-bind*: If no matching service instances can be found by "control tasks" during execution, then the user has to provide different values for the task workflow parameters which are bound to the task workflow (e.g., different dates for flights).
- *Re-configuration*: If the environment changes during execution, e.g., a service is suddenly not available any more, then services processes and service instances need be to re-configured to find an alternative solution. Changes in the environment are identified by "control tasks" in the task workflow.

In the remainder of this section the composition process steps are explained in the context of the business trip example from Section 2.

4.1 Modeling Service Processes

We now provide an MDA-based approach for describing service processes. Service processes represent the functional infrastructure that is offered by providers, and the execution that is provided by the task workflow. For each offered service process its provider publishes a service process description, describing its properties by states and constraints, and its behavior by states and actions. Furthermore, it specifies a set of parameterised queries, referred to as enquires, that select all or a subset of the service instances available and that retrieve the current state of a specific service instance.

Definition 1 ***(Service Process Description)*** *A service process description D is defined as a tuple (S, T, F, E, A, C) defined on a set of states S, actions T (with a set of arcs $F \subseteq (S \times T) \cup (T \times S)$ describing state transitions via actions), and enquiries E. The properties of each instance of this service process are described by a set of attributes A and constraints C. We assume a subset K of A serves as key to identify individual service instances. (Alternately, a dedicated interaction key k may exist for part of the lifecycle.)*

Simple services are a degenerated case of this definition, represented by a Service Process Description with a single action transiting from initial to final state.

Definition 2 (Service Instance) *A service instance i of a given service D is described by the current life cycle state inhabited by i, written as $lcs(i) \subseteq 2^{S \cup T}$ and a value for each attribute in A. Let I_D denote the set of service instances of a given service D.*

Definition 3 ***(Enquiry Service Description)*** *An enquiry is a parameterised query over the current process state of a subset of service instances for some service description D (*generic enquiry*), or for the current process state of a specific service instance (*specific enquiry*). A generic enquiry service description is defined as a parameterised (first order) predicate*

over the set of service attributes A, returning all service instances that satisfy the predicate. A specific enquiry description is parameterised by the interaction key of the specific service instance to be selected. Providing actual parameter values for an enquiry service description *results in a concrete enquiry.*

Example: Let us consider the service process descriptions of two airlines, DeepBlue and PurpleRed. For simplicity, we assume that our two airlines offer only return tickets and quote only the cheapest price for each itinerary. For brevity, we also consider a simplified service process.

Potential service instances (un-booked flight offers) of both airlines are identified by attributes fltNo, departureDate, retFltNo, retDate, and are described by the ticket price. A reserved or booked itinerary is identified by a reservationNo that is unique per airline. The reservation number is returned by the first action in a service process e.g, reserve with DeepBlue or book&pay with PurpleRed. Both airlines offer two enquires: obtainQuote (from, to, depDate, retDate), which return a set of quotes for the specified itinerary, and checkStatus(reservationNo) which returns for the current status of a reservation.

Figures 2 and 3 depict the service process descriptions of airlines DeepBlue and PurpleRed, respectively. The figures show actions and associated state transitions. While airline DeepBlue offers the possibility to make reservations (reserve) and to cancel or confirm the reservation later, which results in a booking that is paid (pay) and used (checkIn), airline PurpleRed offers only instant ticket sales (book&pay). Once a ticket is sold it may be used checkIn or canceled (cancel). Canceled tickets may be entitled to a refund, subject to a deducted cancellation fee (collectRefund).

The life cycle of the service process of our hotel group StarHotels consists of a single action only, makeReservation(quoteNo). The service provider StarHotels offers one enquiry service obtainQuote (fromDate, toDate) returning a set of quotes for the indicated period, each one identified by a unique quoteNo and described by the hotel name and dailyRate.

While flight quotes will be priced in Australian Dollar, hotel quotes will be in Euro.

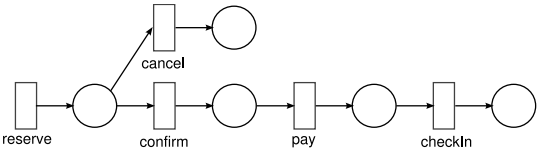

Figure 2: Deepblue Airline Service Process (simplified)

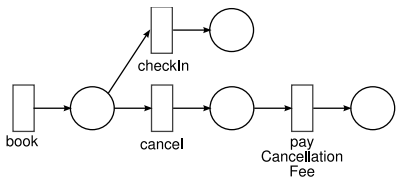

Figure 3: PurpleRed Airline Service Process (simplified)

4.2 Modeling the Task Workflow

The task workflow is the "workhorse" of the approach and serves as the anchor for the process logic and the service matching and instantiation. It is specified by the user on the abstract level of the approach.

Definition 4 ***(Task Workflow Description)*** *A task workflow description W is defined as a tuple $(S_w, T_w, F_w, E_w, K_w, A_w, C_w, O_w, D_w, M_w, b_w, e_w, v_w, s_w)$ defined on a set of states S, actions T (with a set of arcs $F \subseteq (S \times T) \cup (T \times S)$ describing state transitions via actions T_w), and enquiries E. The properties of each instance of this service process are described by a set of attributes A_w and constraints C_w. $K_w \subseteq T_w \times S_w$ indicates a set of* k.o.*-links between activity and states, describing what states (referred to as* k.o. *post states) are entered when an activity completes unsuccessfully. Ordinary links and post states, i.e., those determined by F_w, are called* o.k.*-links and* o.k. *post states, respectively.*

D_w is the set of service process descriptions from which the task workflow is composed.

O_w is a set of abstract service objects, each potentially assigned to a specific service instance of some service process.

$M_w : T_w \rightarrow D_w \times T_w$ associates with each activity an activity of some service process, where if $M_w(t) = (d, t)$, t is an activity of service process description d.

Partial function $b_w : O_w \rightarrow I$ binds each abstract service object to some service instance.

$E_w : O_w \rightarrow 2^{D_w}$ associates with each abstract service object the set of service process descriptions to whose instances the abstract service object may be bound.

Partial function $e_w : O_w \rightarrow I \times D_w$, where for $e_w(o) = (i, d)$, it holds that $d \in E_w(o)$ and $i \in I_d$, associates each abstract service object o to a pair of service instance and service process.

Partial function $v_w : O_w \times I \times A \rightarrow V$ assigns each abstract service object o for each service instance in $e_w(o)$ a value for each attribute. (Such a value corresponds initially to the attribute value of the service instance, but may become "dirty" if the service instance of the service processes involves independently over time, e.g., flight is booked by someone else).

Function $s_w : O_w \rightarrow SOS$ where SOS ={unbound, tentative, committed, canceled, fulfilled} is the set of possible Service Object States.

The workflow activities T_w are distinguished into service usage activities (SUAs), enquiry activities, bridging activities, and control activities. The insertion of bridging and control activities is system supported if data type heterogeneities or critical execution states of the task workflow are identified. A critical execution state might be the begin or end of a milestone (e.g., after obtaining quotes or booking itineraries) when control activities monitor service processes by querying service instances for state changes that need to be handled (e.g., booking an itinerary failed).

Example: The task workflow of our business trip consists of the service process description described above. The user models a simple task workflow similar to a reference process which is refined automatically in the candidate selection step using configuration techniques. Refinement means that modelled activities are refined with subactivities and new activities may be inserted in a consistent way. Consistency criteria using inheritance which are applicable in this step have been investigated in previous work [31]. Figure 4 shows an example for an initial task workflow modelled by the user.

The task workflow has two abstract service objects, flightItinerary and hotelBooking which are not explicitly modelled in Figure 4. Each instance of the task workflow is described by the following attributes: fromCity, toCity, preferredDepDate, preferredRetDate, maxAirFare, minStars, maxHotelCosts, maxOverallCosts, earliestDep, latestDep, earliestReturn, latestReturn, miniumStay,

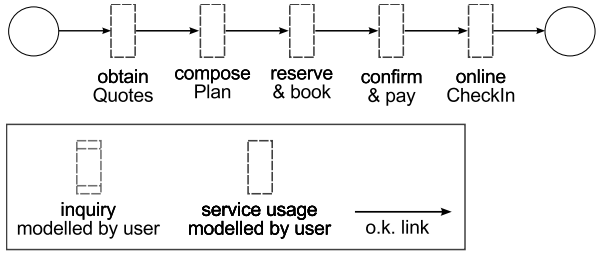

Figure 4: Initial task workflow modelled by the user.

maximumStay, deviateThreshold. The attributes are self explanatory, except deviateThreshold which indicates the percentage in savings which justify choosing an alternative itinerary than that of the preferred dates. Attributes are extracted from user-defined requirements. During execution some service providers may require further attributes which are added dynamically during runtime (e.g., frequent flyer number).

We describe the execution semantics of a task workflow in this section prior to the candidate selection and execution level for better understanding.

Execution Semantics: We define the notion of workflow state, formally capturing the current processing state of a workflow instance (which is referred to as life cycle state) and the workflow execution schedule, determining how the workflow should advance from its current processing state through the execution of service usage activities. As the outcome of service usage activities in the workflow execution schedule is not known in advance, i.e., the booking of a quoted hotel room may be successful or not, the different steps of the execution schedule may be 'guarded, where a guard is expressed by a life cycle state of the workflow instance. The execution of a workflow schedule and the semantics of guards is explained below.

Definition 5 (Workflow State) *The workflow state of a service instance i of a given service D is described by a tuple $< clcs, es >$ consisting of the current life cycle state lcs inhabited by i and the execution schedule $es = \langle s1, \ldots, s_n \rangle$ (which is a list of steps to be executed (started) in order in the future The current life cycle state $clcs$ of the workflow instance is expressed by the states in which the workflow instance currently resides; if an activity is currently executed, it is also part of the $clcs$. Each step s_i $(i = 1 \ldots n)$ of the execution schedule es is a triple $\langle lcs, a, b \rangle$, where lcs is a life cycle state that acts as a guard indicating the step to be executed only if it matches the current life cycle state of the workflow instance, a indicates the activity that is to be invoked if the step is executed, and b determines the actual parameters of the activity invocation.*

The execution schedule, the output of the matchmaking process, is a list of activities with actual parameter bindings. Service execution means calling the individual activity at the head of the execution schedule with the proper parameter binding. If the activity is guarded and the current life cycle state of the workflow instance matches the life cycle state indicated by the guard, the activity is executed. It is ignored otherwise, and the execution schedule is continued with the next step.

Formally, given current lifecycle state clcs and execution schedule $es =< s_1, \ldots, s_n >$ with $s_1 =< lcs_1, a_1, b_1 >$, we have the following situation. If $lcs_1 = \emptyset$ or $lcs_1 = clcs$, then we execute a_1 with b_1 as parameters. If $lcs_1 \neq clcs$, the activity is discarded. In either case, the new execution schedule es' is defined as $es' =< s_2, \ldots, s_n >$ (This description does not include the semantics of control activities which alter the execution schedule).

The current workflow object O_w is omitted if it is understood.

To simplify the execution model, we make the assumption that individual activities in the service process (enquiries and actions) are atomic.

Per the definitions of the previous section, we have an environment e_w (potential bindings) plus instantiated (concrete) query (the latter for optimization purposes) for each abstract service object, plus global constraints given with the service description, plus global workflow constraints C_w, plus global attributes A_w (which must be bound to values).

4.3 Candidate Selection

The candidate selection follows the modelling step and consists of the service and service instance configuration.

During service configuration, services are discovered, matched, and introduced into the task workflow, i.e., the task workflow is modified. The remainder of this paper focuses on service instance configuration and shows only one service configuration step.

Service instance configuration consists of instance discovery and matching. For better understanding of the approach, we first explain briefly the service configuration part with an example, then the service *instance* configuration part and later mediation, matching, and configuration in more detail.

Service Discovery: The identification of relevant service processes is conducted by existing approaches like the ones mentioned in [17]. Here we describe three efforts used in Semantic Web Service discovery that are applicable: First, keyword-based discovery can be applied on a directory, e.g., UDDI, based on the names of the *abstract service objects* defined in a *task workflow description* (cf. Def.4). Second, subsumption-based discovery can be applied by consuming functional requirements defined as *constraints* for each *service usage task* (cf. Def.1), and third, state-based discovery is applicable by using global requirements defined in the task workflow description (cf. Def.4) and service usage task specific constraints like pre- and postconditions (effects) (cf. Def.1).
Example: Abstract service object flightItinerary may be bound to a service instance of either DeepBlue or PurpleRed, and abstract service object hotelBooking to a service instance of StarHotels.

The binding of the abstract service objects to service instances is constrained by the collected user requirements.

Fig. 5 depicts the life cycle of the task workflow of our business trip after the service configuration was executed and the task workflow from Figure 4 was refined and extended. An instance of the task workflow is created by introduced control activity collectTravelReq, which collects the travel requirements from the user and initializes the attributes of the workflow instance accordingly. If additional attributes (e.g., frequent flyer numbers) are required by certain service providers then the control activity takes care of that.

Thereafter, activity obtainQuotes is executed which was refined into three enquiry activities, obtainQuotes_DB, obtainQuotes_PR, obtainQuotes_SH, to obtain quotes from service providers DeepBlue, PurpleRed and StarHotels, respectively. The three subtasks may be executed in any order. As hotel quotes are in Euro,

they are converted by a subsequent bridging activity convertCurrency into Australian Dollar. The bridging activity was also introduced by the configuration task.

Once these activities have been performed, the quotes are matched and selected by the control activity composeTravelPlan.

If a flight itinerary and a hotel quote could be found that meet the specified user requirements, the workflow can proceed with respective bookings.

If the offers cannot meet the user requirements, the workflow is continued by following the *k.o. link* and continued with a control activity that recollects changed user requirements reCollectTravelReq.

For better readability of the Figure, only states to which we explicitly refer to in our examples are named.

The example mentioned before describes the result of the service configuration step. In the remainder, the concepts that result task workflow are discussed.

Service Instance Discovery: The task workflow will usually include a set of enquiry activities that query service providers for available service objects, i.e., instances of service processes, that are considered relevant for the subsequent service matching.
Example: In our business trip setting, which should start on 2nd of July at the earliest and conclude on 18th of July at the latest, only flight itineraries between Adelaide and Vienna whose departure is on 2nd of July or later and whose return flight is on 18th of July or earlier, are relevant.

The first table in Figure 6 depicts e_w(flightItinerary), the potential bindings of abstract service object flightItinerary. The set of potential bindings is determined by the result of executing in our task workflow the enquiry activities obtain_DB_quotes (followed by bridging activity convertCurrency) and obtain_PR_quotes for available flights in the period between 2nd of July (earliest departure) and 18th of July (latest return). The second table in Figure 6 depicts e_w(hotelBooking), the potential bindings of abstract service object hotelBooking. The set of potential bindings is determined by the result of executing in our task workflow enquiry activities obtainQuotes_SH for available hotels in the period between 2nd of July (earliest departure) and 18th of July (latest return).

SP	fltNo	date	retFltNo	retDate	price
DB	DB1	4/Jul	DB2	14/Jul	2.780
DB	DB1	5/Jul	DB2	18/Jul	2.390
DB	DB1	2/Jul	DB2	12/Jul	2.660
PR	PR5	4/Jul	PR6	18/Jul	2.640
PR	PR5	6/Jul	PR6	16/Jul	2.800

quoteId	hotel	availability	dailyRate
13	Uni Lodge	2/Jul - 12/Jul	115
21	BlueDanube	2/Jul - 18/Jul	180
23	Budget Motel	6/Jul - 16/Jul	120
27	Park Inn	2/Jul - 18/Jul	190

SP . . . service provider

Figure 6: Potential bindings for abstract service object flightItinerary, available after obtaining airline quotes, and abstract service object hotelBooking, available after obtaining hotel quotes

Service Mediation: For the semantic mediation, we distinguish between data and behavior mediation. For data mediation, an ontology alignment approach is used as proposed by WSMO [29]. This handles mismatches at the data definition and message exchange protocol level. The handling of heterogeneous business processes is addressed by a catalog of declarative integration patterns defined in previous work [13]. The catalog defines patterns used for the horizontal and vertical integration in Enterprise Application Integration (EAI) and business-to-business (B2B) scenarios. Vertical integration focuses on services in a similar domain and horizontal integration targets services from different domains contributing to a common goal.
Example: A set of airline booking services need to be integrated *vertically* in order to choose the best offer according to some preferences for a specific flight. An airline booking service and a hotel booking service need to be integrated *horizontally* in order to complete a travel booking.

Service Matching: Constraint-based systems, which have been successfully applied to a number of large-scale industrial problems, are potentially useful in the Web service composition context. Previous works, in particular [2, 16], 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, [16] provided an optimized approach to carry out simple checks on Web service attributes. The need for a matchmaking process, that filters Web service candidates based on the functionality a Web service offers, for use within semantic Web based tools 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. Our approach incorporates the semantics (as specified in the abstract workflow, any additional operation specifications, plus functional requirements specified by the user) to generate concrete workflows for any given abstract workflow.

The composition process begins with the design of an abstract workflow. An abstract workflow is a pre-specified control flow specification between abstract service specifications. An abstract service specification is a representation of functional and additional individual requirements that a concrete service has to satisfy. Given an environment of Web services with declarative specifications of their properties such as functionality, semantic service composition is the process of formulating concrete instances of an abstract workflow using the environment such that all the functional, individual and global requirements specified in the abstract workflow are satisfied.
Example: Figure 7 depicts the tentative bindings of abstract service objects FlightItinerary and hotelBooking after service matching and composition and before the respective reservations or bookings have been made.

SP	fltNo	date	retFltNo	retDate	price
DB	DB1	2/Jul	DB2	12/Jul	2.660

quoteId	hotel	availability	dailyRate
21	UniLodge	2/Jul - 12/Jul	135

Figure 7: Tentative bindings of abstract flight and abstract hotel after service matching

Example: Figure 8 depicts the workflow state and execution schedule after service matching. The workflow is in life cycle state { travelPlanComposed }, which is the post state of control activity composeTravelPlan in our task workflow. Two activities are scheduled in accordance with the task workflow description, the reservation of the flight itinerary (unguarded activity reserve_DB) with service provider DeepBlue for the tentative binding of abstract service object flightItinerary

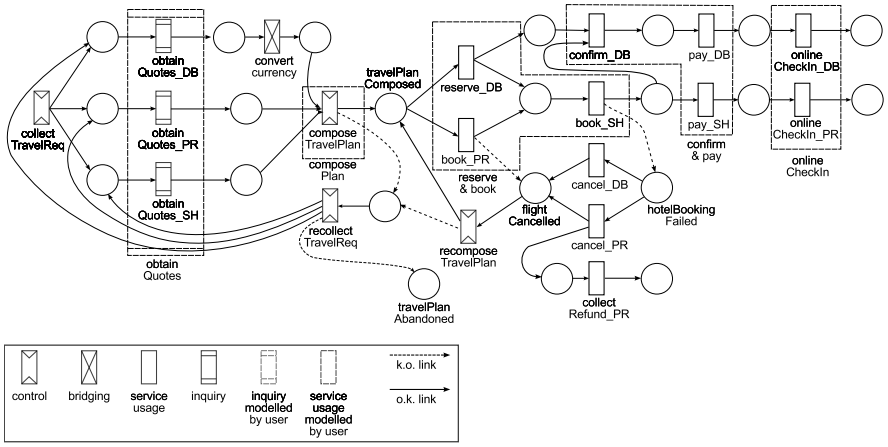

Figure 5: Task Workflow: Business Trip

and the booking (activity guarded activity book_SH) of the hotel with service provider StarHotels according to the tentative binding of abstract service object hotelBooking (cf. Figure 8). The second step of the execution schedule is guarded by life cycle state {hotelBookingFailed} which is the *ko-post state* of activity book_SH. This means that the third step (activity cancel_DB) will be executed only if the requested hotel booking fails once a quoted offer is no longer available. The fourth state flightCancelled is entered after cancel_DB is executed and enables the execution of activity recomposeTravelPlan.

step#	lcs	activity	p.bindings
1		reserve_DB	dept: (DP1,1/Jul), ret: (DP2,11/Jul)
2		book_SH	quoteId: 21
3	{ hbf }	cancel_DB	
4	{ fcld }	rTp	

clcs = { travelPlanComposed }
rTp ... recomposeTravelPlan,
hbf ... hotelBookingFailed
fcld ... flightCancelled

Figure 8: Workflow state with execution schedule after service matching, i.e., after execution of control activity composeTravelPlan

The profile of a service is an advertisement of the capabilities or functionality that the service claims to offer. Its two parts (functional properties or attributes on one hand and invariants on the other) should be satisfied by all the concrete service instances which are advertised using this profile.

Semantic candidate selection is the process of discovering Web services from one or more registries by performing a matchmaking operation on the advertisement of the Web service capabilities against a set of user specified requirements which in general represents the functionality a user expects the Web service to offer.

Definition 6 ***(Service Matchmaking Request)*** *A service matchmaking request R is a set of constraints R_{cons} usually representing the functionality of a Web service that a user of the matchmaking procedure is searching for.*

Definition 7 ***(Constraint Satisfaction Problem)*** *A constraint satisfaction problem (CSP) is defined as a triple $\langle V, D, C \rangle$ where*

- *V is a set of variables $\{v_1, v_2, \ldots, v_n\}$*
- *$domain(v_i)$ denotes the set of allowable values the variable v_i can be assigned with.*
- *D is a set of domains $\{D_1, D_2, \ldots, D_n\}$ such that the $D_i = domain(v_i)$*
- *C is a set of constraints that need to be satisfied for any given assignment to the variables in V.*

A *solution* to a CSP is an assignment of values to variables such that all constraints are satisfied.

Frequently, in modeling complex, component-based systems (hardware or software), this basic definition is extended to to provide individually identified components with a structure defined by a type hierarchy (i.e., object structures), An object's attributes are variables of the CSP, and an object's references (instance variables referencing other objects) also are interpreted as variables in a CSP. However, their domain is special: it is the set of components (or, in typed systems, the set of components of the proper type). This object-oriented constraint view is the modeling approach used by commercial configuration tools and, e.g., in [2], maps naturally to a service composition problem where service instances, activities in the workflow, and data processed by services are considered first-class objects, and the invariants, constraints and pre- and postconditions (guards) associated with services and data objects are constraints on these objects and their values and references (i.e., relationships).

Service Configuration: The service configuration consists of the sequential execution of the composition task (a) "Identification of pairs of service usage tasks and service candidates" and (b) "Request for user feedback for changing the execution order of service usage tasks in the task workflow description". Composition task (a) performs the configuration using the description of previously discovered services, the task workflow description, all requirements (function, non-functional, global requirements), and any necessary and available bridging tasks as inputs. If the task is successful then it produces a list of pairs that assigns for each service usage task a concrete service and the task workflow description. If the composition task does not find a solution, i.e., it fails, then the task workflow needs to be re-designed by the user, i.e., the user has to change the requirements and the configuration will be executed again.

During configuration the order of the steps in the task workflow may need to be changed due to heterogeneous business processes of the discovered services. The change results from a re-ordering caused by the

instantiation of an integration pattern. In this case composition task (b) is executed which requests confirmation for the re-ordering from the user. If the user does not agree with the re-ordering then the configuration task searches for an alternative solution.

Configuration is finished when a concrete service is assigned to each service usage task in the task workflow. It may be executed again if the environment of a task workflow instance is changed during execution and part of the workflow state enters a compensation task.

4.4 Execution

Under certain circumstances, when an action fails and steps have to be retaken, existing objects may have to be wound up, i.e., finish their lifecycle according to their current condition, as part of the compensation activities for the problem.

Example: Figure 9 depicts the execution stage of our workflow instance after the tentative hotel booking at UniLodge hotel from 2/Jul till 12/Jul (cf. Fig. 7) according to quote no. 21 failed due to the quoted offer no longer being available. The workflow instance has entered the *k.o.* post state hotelBookingFailed of activity book_SH. The workflow has entered the compensation phase. The flight reserved with airline DeepBlue needs to be cancelled (activity cancel_DB of step 1) and the travel plan needs to be re-composed. This re-tasking step is represented by control activity recomposeTravelPlan that is scheduled as step no. 2 in the workflow execution schedule after the flight has bee canceled. This control activity will invoke another match-making process and, thereupon, define another workflow execution schedule and enter state travelPlanComposed. If this match-making process fails, the workflow continues with another control activity, recollectTravelReq, in which the user can change the travel requirements or abandons the business trip.

step#	lcs	activity	p.bindings
1	{ hbf }	cancel_DB	quoteId: 21
2	{ fcld }	rTp	

clcs = { hotelBookingFailed }
hbf ... hotelBookingFailed
fcld ... flightCancelled
rTp ... recomposeTravelPlan

Figure 9: Execution stage during compensation phase after tentative hotel booking found unavailable

In some cases, it may be useful to re-task a workflow, i.e., to determine new bindings for abstract service objects, while the compensation phase is still incomplete. In such a case, a *wind-up* copy of the task workflow instance is created that exists concurrently to the current workflow version. The wind-up area may maintain a separate branch until it has reached the end of its lifecycle. A wind-up areas of a task workflow is identified by a set of activities and states in the workflow description.

Example: In our business trip workflow, service usage activity collectRefund_PR together with its prestate and its poststate constitutes a wind-up area. Notice that this area is not explicitly emphasized as such in Fig. 5. The wind-up process maintains the binding of abstract service object flightItinerary such that a refund can be obtained for the canceled flight booking (service usage activity collectRefund_PR). Concurrently, the proper workflow process can be continued by recomposing the travel plan (control activity recomposeTravelPlan) and, thus, associating another flight itinerary (service process instance) to the abstract service object flightItinerary.

5 Conclusion

We have presented a framework for service composition based on conceptual modeling principles, i.e., the use of high-level models of the data and processes involved in the implementation and execution of collaborative service processes. It considers service composition as part of (and result of) a complex design process that in real-world applications has to permit inclusion of user decisions and to execute processes in a distributed, non-atomic environment. The conceptual model serves to describe schema level information and process templates that can be instantiated for the execution of actual services, a distinction that is routine in classic data modeling, but not normally made so far in service composition. By separating out the schema and instance levels, finer distinction between potential service capabilities and actual runtime executability can be achieved. Most importantly, based on this capability, the conceptual models include primitives for modeling decisions taken during the design stage (i.e., the "including control tasks" phase of the process development and execution cycle).

This provides a structured framework for describing how a process can be readjusted at runtime according to information that in real world scenarios is ultimately dependent on execution of particular service instances and therefore not available at static design time, such as the availability of seats on a particular flight at the moment of booking a whole itinerary that was previously planned. The framework enables the composition of service processes under control of the process itself, including the ability to adjust the process if conditions imposed by the environment have changed.

We plan to implement the conceptual model with a similar visual notation as shown in Figure 5 in the DoME[2] meta modelling framework. We extended DoME with Web Service execution capabilities and plan to link the prototype to existing Enterprise Service Bus (ESB) implementations. Using such an infrastructure would allow to integrate existing technologies if provided in Web service form.

References

[1] Michael Adams, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *Proc. OTM*, LNCS 4275, pages 291–308. Springer, 2006.

[2] Patrick Albert, Laurent Henocque, and Mathias Kleiner. Configuration-Based Workflow Composition. In *Proc. ICWS*, pages 285–292. IEEE Press, 2005.

[3] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A Framework for Executing Adaptive Web-Service Processes. *IEEE Software*, 24(6):39–46, 2007.

[4] Steffen Balzer, Thorsten Liebig, and Matthias Wagner. Pitfalls of OWL-S – A practical Semantic Web Use Case. In *Proc. ICSOC*, pages 289–298, 2004.

[5] Boualem Benatallah and Hamid Reza Motahari-Nezhad. ServiceMosaic Project: Modeling, Analysis and Management of Web Services Interactions. In *Proc. APCCM*, volume 53 of *CRPIT Series*, pages 7–9. ACS, 2006.

[2] Domain Modelling Environment (http://www.htc.honeywell.com/dome/)

[6] Eric D. Browne, Michael Schrefl, and James R. Warren. Goal-Focused Self-Modifying Workflow in the Healthcare Domain. In *Proc. HICSS*, 2004.

[7] Marin Dimitrov, Alex Simov, Sebastian Stein, and Mihail Konstantinov. A BPMO Based Semantic Business Process Modelling Environment.

[8] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.

[9] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker. Transforming UML Domain Descriptions into Configuration Knowledge Bases. In *Knowledge Transformation for the Semantic Web*, pages 154–168. IOS Press, 2003.

[10] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, 1998.

[11] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. ODE SWS: A Framework for Designing and Composing Semantic Web Services. *IEEE Intelligent Systems*, 19(4):24–31, 2004.

[12] Roy Grønmo and Michael C. Jaeger. Model-Driven Semantic Web Service Composition. In *Proc. APSEC*, pages 79–86. IEEE Press, 2005.

[13] Georg Grossmann. *Horizontal and Vertical Integration of Object Oriented Information Systems Behaviour*. PhD thesis, School of Computer and Information Science, University of South Australia, 2008.

[14] Georg Grossmann, Michael Schrefl, and Markus Stumptner. Modelling Inter-Process Dependencies with High-Level Business Process Modelling Languages. In *Proc. APCCM*, volume 79 of *CRPIT Series*. ACS, 2008.

[15] Rachid Hamadi, Boualem Benatallah, and Brahim Medjahed. Self-adapting recovery nets for policy-driven exception handling in business processes. *Distributed and Parallel Databases*, 23:1–44, 2008.

[16] Ahlem Ben Hassine, Shigeo Matsubara, and Toru Ishida. A Constraint-Based Approach to Horizontal Web Service Composition. In *Proc. ISWC*, pages 130–143, 2006.

[17] Vipul Kashyap, Christoph Bussler, and Matthew Moran. *The Semantic Web – Semantics for Data and Services on the Web*. Data Centric Systems and Applications (DCSA). Springer, 2008.

[18] Dominik Kuropka, Peter Tröger, Steffen Staab, and Mathias Weske, editors. *Semantic Service Provisioning*. CAT-SWS-1. Springer, 2008.

[19] Freddy Lécué and Alain Léger. Semantic Web Service Composition Based on a Closed World Assumption. In *Proc. ECOWS*, pages 233–242. IEEE Press, 2006.

[20] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *Proc.of WWW Conference*, pages 331–339. ACM Press, 2003.

[21] Peter Mangan and Shazia Sadiq. A Constraint Specification Approach to Building Flexible Workflows. *JRPIT*, 35(1):21–39, 2003.

[22] Wolfgang Mayer, Rajesh Thiagarajan, and Markus Stumptner. Service Composition as Generative Constraint Satisfaction. In *Proc. ICWS*, pages 888–895, 2009.

[23] Sheila McIlraith and Tran Cao Son. Adapting Golog for the Composition of Semantic Web Services. In *Proc. KR*, pages 482–493, 2002.

[24] Martin Necasky and Jaroslav Pokorny. Designing Semantic Web Servicse using Conceptual Model. In *Proc. of ACM SAC*, pages 2243–2247. ACM Press, 2008.

[25] Michael P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[26] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. IJCAI*, pages 1252–1259, 2005.

[27] Günter Preuner, Christian Eichinger, and Michael Schrefl. Static-Dynamic Integration of External Services into Generic Business Processes. In *Proc. ICSNW*, LNCS 3226, pages 263–277, 2004.

[28] M. Reichert, P. Dadam, S. Rinderle-Ma, A. Lanz, R. Pryss, M. Predeschly, J. Kolb, L.T. Ly, M. Jurisch, U. Kreher, and K. Goeser. Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite. In *Proc. BPM Demonstration Track*, CEUR WS Vol.489, 2009.

[29] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubn Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[30] M. Sabou, D. Richards, and S. van Splunter. An experience report on using DAML-S. In *Proc. of WWW workshop on E-Services and the Semantic Web (ESSW)*, 2003.

[31] Michael Schrefl and Markus Stumptner. Behavior-consistent Specialization of Object Life Cycles. *ACM TOSEM*, 11(1):92–148, 2002.

[32] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[33] Rajesh Thiagarajan and Markus Stumptner. Service Composition With Consistency-based Matchmaking: A CSP-based Approach. In *Proc. ECOWS*, pages 23–32, 2007.

[34] Michele Trainotti et al. ASTRO: Supporting Composition and Execution of Web Services. In *Proc. ICSOC*, pages 495–501, 2005.

[35] Kees van Hee et al. Adaptive Workflows for Healthcare Information Systems. In *Proc. BPM Workshops*, LNCS 4928, pages 359–370. Springer, 2008.

[36] Bin Yu, Chen Zhang, and Yang Zhao. Transform from Models to Service Description Based on MDA. In *Proc IEEE APSCC*, pages 605–608. IEEE Press, 2006.