Web Service Composition

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Ingo Weber
NICTA
Sydney, Australia
ingo.weber@nicta.com.au

Synonyms and Keywords

Business process modeling, service adaptation, service-oriented architectures, semantic web service composition, planning.

Glossary

Web Service: Self-contained software module available via a network.
SOA: Service-oriented architecture, a software architecture based on Web services.
BP: A Business Process "consists of a set of activities that are performed in coordination in an organizational and technical environment."[36]
BPM: Business Process Management "includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes."[36]
Composition: A combination of web services achieving a higher-value service.
BPEL: Business process execution language, a programming language for creating compositions.
BPMN: Business process model and notation, a high-level notation and semantics for compositions.
Manual composition: The process of creating a composition by hand.
Automatic composition: The process of creating a composition based on a description of what it should achieve.
Planning: An area of Artificial Intelligence aiming at equipping computers with general problem solving capabilities.

Definition of the Subject

The prevalent architectural model of modern enterprise application software and other distributed systems is Service-Oriented Architecture (SOA): applications are split into reusable blocks of functionality – the *services* – which are exposed to respective consumers. A Web service is a "self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application" [24]. Examples for publicly available Web services include US National Weather's weather service[1] or Amazon's book search service[2]. An important aspect of designing services is to assure they can be usefully applied in various contexts [7]. As such, Web service *composition* refers to both the activity and the result of combining multiple Web services into a higher-value service.

Two flavours of Web service compositions are commonly distinguished: orchestration and choreography. An *orchestration* of Web services is the "inside-out view" [26] on an implemented business process of an organization – while containing the information how to interact with business partners, it also describes how and when to invoke Web services of the organization itself. In contrast, a *choreography* "defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state" [33]. Orchestrations are scripts that are detailed enough to be actively executed, typically implementing a composition from the viewpoint of one organization. In contrast, choreographies are global contracts between participants of equal importance; adherence to the contract can be passively observed, but the contract cannot be actively executed.

Introduction

In this chapter, we describe the main aspects and topics surrounding Web service composition. The main distinction we make is whether a composition is created *manually*, by a human modelling the composition in a suitable language, or *automatically*, by a program that accepts a declarative description of the composition-to-be as well as the available services, upon which the program combines a suitable subset of the available services into a composition. Various languages have been proposed for manual composition, ranging from programming-like languages expressed in XML, such as the Web Services Business Process Execution Language

[1] http://graphical.weather.gov/xml/
[2] http://soap.amazon.com/schemas2/AmazonWebServices.wsdl

(WS-BPEL) [22], over graphical languages such as Business Process Model and Notation (BPMN) [23], to domain expert-focused languages [35]. We give an overview of the main approaches and explain how and where they are applied. Automatic composition frequently builds on AI Planning. We overview the foundations, and explain how these techniques are applied to automatically generate orchestrations.

Key Points

Web service composition is a software-engineering technology allowing to flexibly recombine pre-existing functionalities, across physical and organizational boundaries. Automation techniques facilitate this task by allowing to conveniently find relevant functionalities in large repositories, and by pre-composing them in sensible ways. In the present article, we provide a brief overview of the history and current state of this technology, as an area of industrial application as well as academic research. We highlight its use in *Business Process Management (BPM)*, an important application area of Web service composition. The key points we make are:

(i) **Background:** We explain how WSC was initially conceived, and how the composition languages in use today were designed. We outline the academic sources of the main automation techniques.

(ii) **WSC languages:** We overview two of the main languages for defining Web service compositions, *BPEL* and *BPMN*. We summarize the key applications of this methodology, illustrating it with the use of BPMN at SAP.

(iii) **Automation techniques:** We describe two of the main technologies allowing to automatically pre-compose Web services, both based on *AI Planning* technology. We summarize the key applications, and outline a concrete application we developed at SAP.

Historical Background

Web Service Composition

The main drivers behind Web services and their composition are two-fold. On the one hand, the increasing development of enterprise software led to the need for integrating various systems with each other. For instance, making information from a warehouse solution available to a production system, and in turn, linking the latter to the financial planning and reporting system of a company can lead to increased efficiency throughout an organization. However, traditional systems were restricted in the integration technology; e.g., in Java, remote procedure calls could only be achieved between two systems using the same Java version. Web service technology provides a standardized mechanism to interact with remote system components over standard networks, like the Internet. This further allows code reuse on a functional level: functionality available as a service can be reused without having to consider the underlying technology or involving the original developers of the service. The described functionality is also advertised in a standardized way, and consumers of the service can develop their systems against the advertised description.

On the other hand, proprietary systems were often perceived as inhibiting quick changes: the speed with which business demands change is ever-increasing; the speed with which traditional implementations of business processes could be changed did not match this increased pace [10]. This situation called for a shift in the way business applications were implemented, in order to support fast changes in business process implementations or allowing the swift implementation of new processes. Therefore, business process models are nowadays implemented as Web service compositions: rather than changing proprietary systems and integrating them in different ways, the idea is to combine existing blocks of functionality – the services – in novel ways.

Planning for Web Service Composition

The dominating paradigm towards automation of Web service composition, i.e., towards technology that selects and pre-composes Web services based on a high-level specification of the user requirement (that the composed service should accomplish), is AI Planning. That field has its roots at the dawn of *Artificial Intelligence (AI)*, where it was initially conceived having in mind to achieve the flexible problem-solving capabilities of humans. Planning tools are aimed at solving not just one specific problem, but at solving *all* problems, so long as they can be described in a suitable declarative input language.

Concretely, planning problems are described in terms of an *initial state*, a *goal*, and a set of possible *actions*. Given this input, the planning tool automatically derives a *plan*, a schedule of actions transforming the initial state into a state that satisfies the goal. A multitude of variants of this kind of problem have been investigated. For an overview of the area, we recommend the book by Ghallab et al. [8].

Seminal work on automization in WSC [21, 30] viewed Web services as transformations on states, and viewed user requirements as pairs of *preconditions* ("what I got") and *postconditions* ("what I want"). In the infamous "Virtual Travel Agency" example (which is probably not the most relevant application of automated WSC, but serves to illustrate the idea), the Web services implement transactions booking a train or flight ticket, the precondition describes the user's travel preferences, and the postcondition requires

an appropriate journey to be booked. The correspondence to planning is immediate: the Web services are the actions, the user requirement precondition is the initial state, and the user requirement postcondition is the goal. This correspondence has formed the basis of most work in this area, since the early 2000s. A main prerequisite is that the Web services come with a suitable planning-like description of what they do (also a precondition/postcondition pair, in most cases). These descriptions are referred to as *semantic annotations*, and connect automated WSC to *Semantic Web* research. In particular, the most wide-spread semantic Web language, *OWL* [37], has a module *OWL-S* [5] allowing to annotate Web services in this way.

Web Service Composition Languages

As outlined in the introduction, the two main composition languages covered in this chapter are WS-BPEL and BPMN. A composition language is defined by its syntax – its appearance, i.e., XML expressions (BPEL) or graphical symbols (BPMN) – as well as its execution semantics – how combinations of syntax elements translate to process behavior, e.g., a `sequence` in BPEL means that the contained activities are executed in sequential order.

BPEL

BPEL is primarily a block-structured language, i.e., control flow is defined by nesting actions into control flow constructs such as `sequence` mentioned above. We explain the main elements using the example of a phone line order process given in Listing 1. Control flow elements include `if / elseif / else` for conditional execution (see around line 6 in Listing 1, the check if the order comes from an existing customer), `flow` for parallelism (concurrent updating of the provisioning system and order confirmation, around line 13), and `while / forEach / repeatUntil` for repetition. The core activity types around which control flow is structured are `assign` for data manipulation in variables (copying the order information to the internal format, line 4), `invoke` for calling Web services (e.g., checking the customer status, line 5), `receive` for incoming Web service calls (line 3), and `reply` for replying to received calls (order confirmation or rejection, line 16 or 19).

Listing 1: Phone Line Order Process from the telecommunications domain in simplified BPEL.

```
1 <process name="PhoneLineOrderProcess">
2   <sequence>
3     <receive name="ReceiveNewOrder" />
4     <assign name="CopyInformation">
5     <invoke name="CheckCustomerStatus">
6     <if>
7       <condition name="newCustomer"/>
8       <invoke name="CreateNewCustomer">
9     </if>
10    <invoke name="CheckOrder">
11    <if>
12      <condition name="compatibleOrder"/>
13      <flow>
14        <invoke name
15          ="UpdateProvisioningSystems">
16        <reply name="OrderConfirmation">
17      </flow>
18      <else>
19        <reply name="OrderRejection">
20      </else>
21    </if>
22  </sequence>
23 </process>
```

Besides the block-structured parts of BPEL, the `flow` element may be enriched with `links` inducing partial ordering: if there is a link from activity A to activity B, A's execution needs to be completed before B is started. Complex conditions can guide how an activity should behave that is the target of multiple links.

BPEL includes many more elements – such as partner links, event-based behavior, fault handling, and compensation – not discussed here for brevity. Due to its XML-based syntax and close proximity to programming models, BPEL adoption has focused on a technical integration layer between Web service-based systems [32].

BPMN

BPMN in contrast is a language with a graphical syntax, which is said to appeal more to business users. Fig. 1 shows the sample process from Listing 1 in BPMN.

BPMN distinguishes between the main element types of activity (rectangle with rounded corners), event (circle), and gateway (diamond), as well as certain arrow types for expressing control and data flow. Basic activities just contain a textual label, like "Check Customer Status" in Fig. 1.

Start events have a single, thin line surrounding them (e.g., "New Order"), intermediate events a double thin line (e.g., "Reject Order"), and end events a single, thick line (bottom right corner of Fig. 1). The respective meaning of an event is partly expressed by the contained symbol, if any – e.g., the white letter symbol in the shown start event means receiving a message, whereas a black letter symbol, e.g., in the intermediate events in Fig. 1, indicates sending a message. Other symbols (not shown here) include time-related events, errors, compensation, and many more.

Gateway types are distinguished in a similar fashion: a "+" indicates a parallel split/join gateway, an "X" inside a diamond refers to an exclusive-or (XOR) split/join. The former can be seen in Fig. 1 before (split) and after (join) "Confirm Order" / "Update Provisioning Systems". XOR gateways are used, e.g., around the conditional activity "Create New Customer Record" in Fig. 1. The outgoing arrows from the XOR split indicate conditional execution (arrow starting

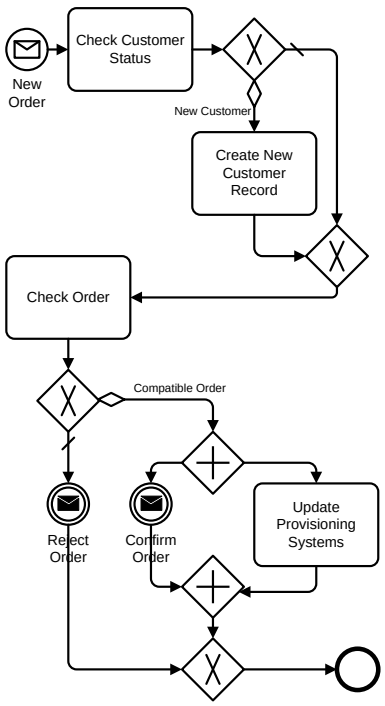

Figure 1: Phone Line Order Process from the telecommunications domain in BPMN.

with a diamond and condition "New Customer") or default (arrow start crossed by a diagonal line). Additional gateway types in BPMN (not shown here) are inclusive-or, complex, and (parallel) event-based.

BPMN also contains elements for modeling process parts in separate spheres of control, choreographies, exception handling, links between several models, and many more details left out here for brevity.

BPMN for Web service composition. Being a graphical, generic business process modeling language, BPMN can be used for many purposes, such as documenting processes or training new employees. However, BPMN is also used for modeling Web service compositions. While this usage was limited in BPMN v1.1 by partially unclear execution semantics, this problem has been overcome in v2.0. Yet, graphical models by themselves do not contain enough details to be an executable as such. Missing information include the Web service location (URL), which operation to call, what data to send in which format, etc. Tool vendors using BPMN for Web service composition allow the specification of these details often in proprietary form, e.g., as properties of activities. Examples of such tools include SAP NetWeaver BPM Process Composer[3], Intalio|BPM Designer[4], and many more.

[3] `http://scn.sap.com/community/bpm`

[4] `http://www.intalio.com/bpms/designer`

Automation Techniques

We now consider planning-based techniques for automated WSC in some detail. A multitude of approaches exist (e.g., [21, 30, 6, 16, 29, 18, 13, 15]), which differ widely in intention, scope, and underlying formalisms. A major distinction line is that between (A) *planning for service chaining* and (B) *planning for service interactions*. (A), but not (B), simplifies the automated WSC challenge – which is essentially a form of automated programming and thus quite difficult – by viewing Web services as one-shot applications, taking into account their input/output typing and high-level properties (like available credit), but ignoring the technical details of interacting with them. Thus (A) provides only a composition template, pre-selecting and arranging a subset of relevant Web services. (B), by contrast, delivers an executable software artifact, i.e., an orchestration. Its disadvantages are computational (practical scaling is much worse than in the simplified variant), as well as in modeling because delivering executable software requires a very fine-grained specification of the user requirement.

In what follows, we mainly focus on (A), because it is easier to comprehend in the limited space, and has relevant applications in BPM. We remark that one can combine both approaches naturally, using (A) as a pre-process to (B). This yields much smaller input for (B), along with additional information how the relevant Web services could be combined, and thereby helps to address both (B)'s scalability issues and modeling requirements [4].

Foundations

To make matters concrete, we describe a simple planning formalism, *planning with finite-domain variables* [11]. A *planning task* is a tuple (X, I, G, A). X is a finite set of *state variables*, where each $x \in X$ is associated with a finite domain D_x. A *partial state* over X is a function s on a subset X_s of X, so that $s(x) \in D_x$ for all $x \in X_s$; s is a *state* if $X_s = X$. The *initial state* I is a state. The *goal* G is a partial state. A is a finite set of *actions*. Each $a \in A$ is a pair $a = (\mathrm{pre}_a, \mathrm{eff}_a)$ of partial states, called its *precondition* and *effect*. Partial states are identified with sets of variable-value pairs, referred to as *facts*. The *state space* of the task is the directed graph whose vertices are all states over X, with an arc (s, s') iff there exists $a \in A$ such that $\mathrm{pre}_a \subseteq s$, $\mathrm{eff}_a \subseteq s'$, and $s(x) = s'(x)$ for all $x \in X \setminus X_{\mathrm{eff}_a}$. A *plan* is a path in the state space, leading from I to a state s with $G \subseteq s$.[5]

For example, say we have a variable for the status of a flight booking, $Pending$ vs. $Confirmed$. The book-

[5] Note that the size of the state space is exponential in the number of variables. Thus this planning language, despite its simplicity, allows to describe compactly a large number of possibilities; deciding whether or not there exists a plan is **PSPACE**-complete.

Action name	precondition	effect
Create CQ		(CQ.approval:necessary OR CQ.approval:notNecessary) AND CQ.acceptance:notAccepted AND CQ.archiving:notArchived AND CQ.submission:notSubmitted
CQ Approval	CQ.approval:necessary	CQ.approval:approved OR CQ.approval:rejected
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived

Figure 2: A SAM-like example, modeling the behavior of "customer quotes" CQ.

ing is currently pending, we wish it to be confirmed, and the only action is a Web service confirming the booking. In the above formalism, this can be modeled as follows. $X = \{\mathit{flightStatus}\}$ with $D_{\mathit{flightStatus}} = \{\mathit{Pending}, \mathit{Confirmed}\}$; $I = \{(\mathit{flightStatus}, \mathit{Pending})\}$; $G = \{(\mathit{flightStatus}, \mathit{Confirmed})\}$; A contains a single action taking the form $(\{(\mathit{flightStatus}, \mathit{Pending})\}, \{(\mathit{flightStatus}, \mathit{Confirmed})\})$ where $\{(\mathit{flightStatus}, \mathit{Pending})\}$ is the precondition and $\{(\mathit{flightStatus}, \mathit{Confirmed})\}$ is the effect. The state space contains the two vertices $\{(\mathit{flightStatus}, \mathit{Pending})\}$ and $\{(\mathit{flightStatus}, \mathit{Confirmed})\}$, the only arc going from the former to the latter. The plan traverses that arc, confirming the flight.

When applying this form of planning to WSC, each Web service is modeled as an action, so we describe its "precondition" and "effect" using the above structures. This allows to express the service's input and output behavior, i.e., the typing of the respective service parameters. As a simple example, the service input may be a customer-data object, and the output may be a reservation object. Further, one can express additional prerequisites or consequences the service may have. For example (presuming a richer planning language dealing with numeric state variables), a precondition "$\mathit{credit} \geq 500$" may require the sufficient availability of money, and an effect "$\mathit{credit} := \mathit{credit} - 500$" may reduce the available money by that amount.

Planning is computationally hard even in its simplest forms, however research has come up with a range of approaches that tend to be effective in practice (e.g. [14, 31]). Provided that the plans to be created are not too large (around a few dozen actions), they are typically found within seconds (e.g. [13, 15]). On the downside, plans here are not executable software artifacts. They disregard the order of interactions required for communicating with the services (making a reservation typically involves several steps), and they disregard the underlying data structures (what exactly is a "customer-data object"?). Given this, in the typical application scenario, a human user is responsible for the overall design, and uses the planning facility for easing the task of selecting and arranging a useful subset of services from a large services database or the internet (e.g. [1, 15]).

In difference to planning for service chaining, planning for service interactions (e.g., [28, 3, 4, 19, 2]) has the ambition to create executable software. Each Web service is modeled as a state transition system, specifying its interaction behavior in terms of a set of different internal states of the service, along with transitions associated with input and output objects. The task is to automatically generate a new transition system, implementing a *controller* that interacts with the given services in a way satisfying a *composition requirement*. The input transition systems correspond closely to BPEL abstract processes, and thus are easy to come by in practice. The difficult issue regarding modeling is the composition requirement, which must be provided by the user desiring to create the process, and which can become rather complex since it must be sufficiently precise to describe the desired software.

Modeling also is a crucial point in the practice of planning for service chaining. Although the annotations needed are more light-weight than in planning for service interactions, the modeling overhead can be prohibitive. In our own work at SAP [15], we showed that, sometimes, it is possible to re-use pre-existing models of software behavior, reducing that overhead dramatically. We now consider this approach in some detail, as an example.

Illustrative Example

SAP widely employs model-driven software engineering. In particular, *Status and Action Management (SAM)* models over 400 business objects (BOs). BOs are software objects corresponding to common business scenarios, like a customer quotes or a sales order. SAM models these complex objects at the business-level of abstraction. Each BO is described using a set of finite-domain status variables, and a set of actions. Each action is specified in terms of a precondition and an effect, describing which status values are required in order to execute the action, respectively how the action may change these values. For illustration, Figure 2 gives a SAM-like model for a BO called "customer quote

(CQ)".[6] In the figure, by "CQ.X:Y" we denote the atomic proposition stating that variable X of the customer query has value Y.

The original purpose of SAM is code generation: code skeletons check if preconditions are fulfilled at runtime, and update the status variables accordingly. Despite this difference in purpose, clearly SAM corresponds closely to planning with finite-domain variables. Indeed, the only differences are that SAM preconditions are logical formulas including "OR" and "NOT" connectives, and that SAM action effects can be non-deterministic, i.e., have several possible outcomes. Both have long been subject of research in AI Planning, and in our work we adapted a wide-spread planning technique [14] to deal with SAM.

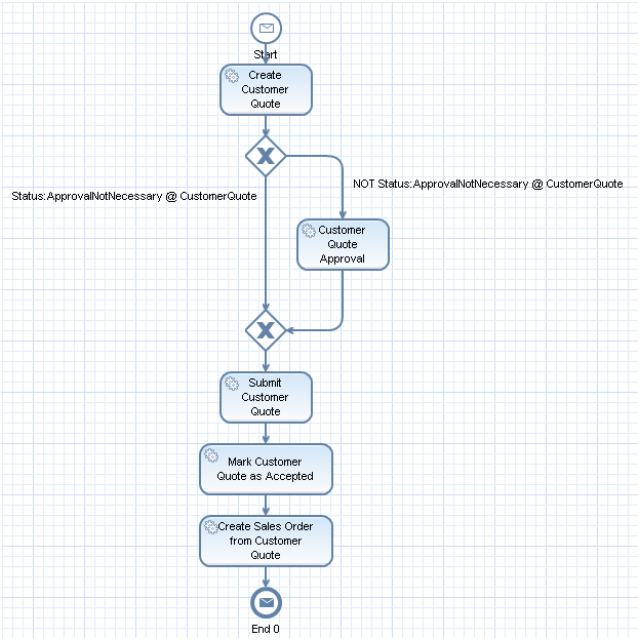

Figure 3: Screenshot of the SAP NetWeaver BPM Process Composer with an automatically composed process of five non-deterministic services.)

The planning functionality is implemented as a prototypical research extension to the SAP NetWeaver BPM Process Composer. In that modeling environment, business users can specify the requirement on a to-be-composed process (typically a variation of a standard process), in terms of changes to SAM status variables, i.e., essentially in their own language. The status values are specified using simple dropdown menus. Pushing a button invokes the planner, which returns a BPMN template for the new process. Figure 3 shows that template, for the example from Figure 2. Note the non-deterministic effect of the create operation (top): approval may or may not be necessary. Depending on which is the case, an additional approval step is included. That step also has two possible outcomes. The process continues only in the positive case. For the negative case, exception handling is needed, and since SAM does not contain information about how that should be done, the planner cannot compose it and leaves this part of the process unspecified. The template is also incomplete in that not every customer quote should be approved, submitted, etc. straight after being created.

[6]For confidentiality reasons, the shown object and model are artificial, i.e., they are *not* contained in SAM as used at SAP.

Key Applications

As outlined, planning for service chaining can be used to help with the generation of new or changed process in BPM. This is important because changes are frequent in dynamic markets, while the people responsible for adapting the processes – business experts – are not familiar with the underlying IT infrastructure. Designing executable processes requires intensive communication between business experts and IT experts, incurring significant costs for human labor and increased time-to-market. Planning, as in our application based on SAM at SAP, helps business experts to come up with a process close to the IT infrastructure, and thus helps to bridge the expertise gap more efficiently.

Another key application of planning for service chaining is the control of computing and network resources. With the advent of cloud computing, changing the state of resources or acquiring new resources can be achieved by invoking Web services – see e.g., Amazon Web Services[7]. Planning can then be used to bring a system's structure from the current state to a more desired state, e.g., [17, 12, 9, 34].

Planning for service interactions has been successfully applied in a variety of areas, including electronic commerce [20] and electronic government [27].

Future Directions

We highlight a few of the current trends in WSC that we personally consider important. On the side of WSC languages, a fairly recent development is the release of BPMN 2.0, including a profile for specifying choreography models. Another trend is end-user-focused WSC [35], i.e., simplifying the (manual) composition method to a point where business users can code compositions for their personal needs themselves. Last but not least, we want to mention techniques for looser coupling of compositions to Web services. This is particularly interesting when available Web services follow the REST paradigm. Composition methods for REST-based services include adapting BPEL [25], or custom approaches like [38], where REST services include the relevant fragments of the composition in messages returned to the client. Fot automated WSC, the challenge essentially is to keep of

[7]`http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/`

with these developments, since every distinct WSC framework has different prerequisites and requirements on the automation support.

Cross-references

Will be filled in later.

Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Parts of this chapter are based on other publications of the authors, as cited in the text.

References

[1] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava. Synthy: A system for end to end composition of web services. *J. Web Semantics*, 3(4), 2005.

[2] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner. Control Flow Requirements for Automated Service Composition. In *Proc. of the IEEE International Conference on Web Services (ICWS09)*, 2009.

[3] P. Bertoli, M. Pistore, and P. Traverso. Automated web service composition by on-the-fly belief space search. In *16th International Conference on Automated Planning and Scheduling (ICAPS-06)*, 2006.

[4] Piergiorgio Bertoli, Joerg Hoffmann, Freddy Lecue, and Marco Pistore. Integrating discovery and automated composition: from semantic requirements to executable code. In *Proceedings of the IEEE 2007 International Conference on Web Services (ICWS'07)*, 2007.

[5] Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara, and David Martin (ed.). *OWL-S: Semantic Markup for Web Services. OWL-S 1.1*. http://www.daml.org/services/owl-s/1.1/, November 2004. Version 1.1.

[6] Ion Constantinescu, Boi Faltings, and Walter Binder. Large scale, type-compatible service composition. In *2nd International Conference on Web Services (ICWS-04)*, pages 506–513, 2004.

[7] Thomas Erl. *Service-Oriented Architecture: Principles of Service Design*. Prentice Hall, 2007.

[8] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann/Elsevier, 2004.

[9] Sebastian Hagen and Alfons Kemper. Model-based planning for state-related changes to infrastructure and software as a service instances in large data centers. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 11–18, Washington, DC, USA, 2010. IEEE Computer Society.

[10] Randy Heffner, Bobby Cameron, and Kimberly Dowling. Your strategic SOA platform vision crafting your architectural evolution to service-oriented architecture. Technical report, Forrester Research, Trends, 29 March 2005.

[11] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[12] Herry Herry, Paul Anderson, and Gerhard Wickler. Automated planning for configuration changes. In *LISA'11: Large Installation System Administration Conference*, 2011.

[13] Jörg Hoffmann, Piergiorgio Bertoli, Malte Helmert, and Marco Pistore. Message-based web service composition, integrity constraints, and planning under uncertainty: A new connection. *J. Artificial Intelligence Research*, 35:49–117, 2009.

[14] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artificial Intelligence Research*, 14:253–302, 2001.

[15] Jörg Hoffmann, Ingo Weber, and Frank Michael Kraft. SAP speaks PDDL: Exploiting a software-engineering model for planning in business process management. *J. Artificial Intelligence Research*, 44:587–632, 2012.

[16] U. Kuter, E. Sirin, D. Nau, B. Parsia, and J. Hendler. Information gathering during planning for web service composition. *J. Web Semantics*, 3(2-3):183–205, 2005.

[17] Feng Liu, Vitalian Danciu, and Pavlo Kerestey. A framework for automated fault recovery planning in large-scale virtualized infrastructures. In *MACE 2010, LNCS 6473*, pages 113–123, 2010.

[18] Zhen Liu, Anand Ranganathan, and Anton Riabov. A planning approach for message-oriented semantic web service composition. In *22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, 2007.

[19] A. Marconi and M. Pistore. Synthesis and composition of web services. In *Formal Methods for Web Services*, pages 89–157. Springer Berlin / Heidelberg, 2009.

[20] A. Marconi, M. Pistore, and P. Traverso. Automated Web Service Composition at Work: the Amazon/MPS Case Study. In *Proc. of IEEE International Conference on Web Services (ICWS'07)*, 2007.

[21] Sheila McIlraith and Tran Cao Son. Adapting Golog for composition of semantic Web services. In *Proc. of the 8th Int. Conf. on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France*, 2002.

[22] OASIS. *Web Services Business Process Execution Language V 2.0*, April 2007.

[23] OMG. Business Process Model and Notation – BPMN 2.0. http://www.omg.org/spec/BPMN/2.0/, 2011.

[24] Mike Papazoglou. *Web Services: Principles and Technology*. Prentice Hall, 2007.

[25] Cesare Pautasso. Restful web service composition with bpel for rest. *Data Knowl. Eng.*, 68(9):851–866, 2009.

[26] C. Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, 2003.

[27] M. Pistore, P. Braghieri, P. Bertoli, A. Biscaglia, A. Marconi, S. Pintarelli, and M. Trainotti. ASTRO: supporting the composition of distributed business processes in the e-government domain. In *At Your Service: Service-Oriented Computing from an EU Perspective*, 2008.

[28] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, 2005.

[29] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite BPEL4WS web services. In *3rd IEEE International Conference on Web Services (ICWS-05)*, 2005.

[30] Shankar Ponnekanti and Armando Fox. SWORD: A Developer Toolkit for Web Service Composition. In *Proceedings of the 11th International WORLD WIDE WEB CONFERENCE – WWW2002*, 2002.

[31] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artificial Intelligence Research*, 39:127–177, 2010.

[32] Terry Schurter. BPM State of the Nation 2009. bpm.com, `http://www.bpm.com/bpm-state-of-the-nation-2009.html`, accessed 25/11/2009, 2009.

[33] W3C. *Web Services Architecture*.

[34] Ingo Weber, Hiroshi Wada, Alan Fekete, Anna Liu, and Len Bass. Automatic undo for cloud management via ai planning. In *Hotdep'12*, Oct 2012.

[35] Ingo Weber, Hye young Paik, and Boualem Benatallah. Forms-based service composition. In *ICSOC'11: 9th International Conference on Service Oriented Computing, short paper*, Paphos, Cyprus, Dec 2011.

[36] Mathias Weske. *Business process management: concepts, languages, architectures*. Springer Verlag, Berlin, Heidelberg, 2007.

[37] World Wide Web Consortium (W3C). *Web Ontology Language (OWL)*. W3C Recommendation 10 February 2004.

[38] Xiwei Xu, Liming Zhu, Udo Kannengiesser, and Yan Liu. An architectural style for process-intensive web information systems. In *WISE'10: Proceedings of the 11th international conference on Web information systems engineering*, pages 534–547, Berlin, Heidelberg, 2010. Springer-Verlag.