MOVE: A Generic Service Composition Framework for Service Oriented Architectures

Conference Paper · July 2009
DOI: 10.1109/CEC.2009.56 · Source: DBLP

CITATIONS
3

READS
14

2 authors, including:

Jürgen Dorn
TU Wien

71 PUBLICATIONS 484 CITATIONS

Available from: Jürgen Dorn
Retrieved on: 15 September 2016
MOVE: a generic service composition framework for Service Oriented Architectures

Albert Rainer  
E-Commerce Competence Center  
Vienna, Austria  
albert.rainer@ec3.at

Jürgen Dorn  
Institute of Software Technology and Interactive Systems  
Vienna University of Technology  
Vienna, Austria  
juergen.dorn@ec.tuwien.ac.at

Abstract

One main characteristic of virtual organizations are short-term collaborations between business partners to provide efficient and individualized services to customers. The MOVE project targets at a methodology and a software framework to support such flexible collaborations based on process oriented design and communication by Web services. MOVE’s framework supports the graphical design and verification of business processes, the execution and supervision of processes in transaction-oriented environment, and the dynamic composition and optimization of processes. A business process may be composed from a set of Web services, deployed itself as Web service and executed in the framework. The composition of processes from Web services is implemented with methods from AI-planning. We map Web service descriptions, domain knowledge and customer requests into the input language of AI-planners. We show the performance of our program and give some implementation details. Finally we conclude with some remarks about further research.

1. Introduction

The rapid creation, configuration, and adaptation of virtual organizations and inter-organizational business processes are major issues for companies and their ability to react to changes in the market. In order to support this rapidness tools may be employed that help in the configuration process and can generate suitable and (possibly) optimal solutions. Those tools are typically stemming from the AI-domain, such as constraint satisfaction solvers for proper configuration of architectures, or planners for business process design. In this work we are considering the latter case and will apply planning techniques in order to create business processes from a set of atomic services.

In the most basic sense, planning is the task of finding a sequence of actions that will achieve a given goal. From this definition one can conclude that there is some equivalence between planning on the one side and modeling respectively execution of business processes on the other side. Both deal with action sequences that transform the initial situation into a desired final state. The Web services challenge follows a similar pattern and thus we consider methods from AI-planning suitable to apply to this challenge.

First, the composition framework and possible use cases are presented. After that, we discuss issues regarding the mapping of the Web services challenge to the planning domain and we show preliminary results of our approach. Finally we conclude hints for further research topics.

2. Service Composition Framework

Service composition is mostly connected to the task of constructing some kind of business process or workflow, i.e. of finding an appropriate sequence of services that will achieve some goal. While this is obviously the major use case, one can distinguish other use cases as well that have high relevance in connection to a composition framework, either at modeling time or at runtime. The following list gives some examples of additional use cases that extend the basic use case:

- Architecture: In contrast to business processes and workflows, this use-case aims to configure a proper and optimized architecture out of a number of services and components. As an example, the Service Component Architecture (SCA) is a means that allows to create and configure an architecture from basic building blocks1.
- Policies: Guide the composition process with business rules, constraints and preferences. One can distinguish between hard and soft constraints; while the former must not be violated, a violation of the latter is acceptable but some “penalty fee” has to be paid. For instance, a hard constraint could be that two services (say Order-product and Pay-product) must belong to the same business partner. As an example for soft constraints, price, time consumption, or reliability of a service may represent the “penalty fee”.
- Composition from Composites: This refers to the ability to build business processes (or architectures) not solely

1. Specifications of SCA and related standards are maintained by Osoa (www.osoa.org/).
from atomic building blocks but also from larger structures. As an example, skeleton processes can contain a number of pre-configured services with gaps that are filled with proper services during the composition task.

- Verification/Validation: Set of engineering techniques to evaluate the quality of systems. Amongst these techniques we consider testing, model checking and other types of system analysis (such as performance analysis) as important features that help to build correctly defined models and reliable systems.

- Reactive planning: At runtime of business process, certain events generated from the environment may lead to re-planning and adaptations of current plans. As an example, when a service does not respond to a request find another service as a substitution.

- Scheduling: In contrast to planning, scheduling deals with proper timing of process steps and utilization of scarce resources.

The global view of the composition framework as displayed in Figure 1 shows that composition can be considered as a highly generic task; it takes input from various sources and generates output for various purposes using a number of appropriate tools for this transformation process. We consider the output being always some kind of model which is subsequently going to be processed by a suitable interpreter; to name a few, a workflow engine, a SOA runtime engine, or a simulation and analyzing software are examples of possible interpreters.

In this framework service descriptions are considered to be at hand in a service repository. Such a repository can be as simple as a flat list of files, or it can be some sophisticated XML-database that holds WSDL documents and other artifacts. The Web service challenge 09 maintains all services in a single WSDL document, therefore this document serves as the repository. Additional input to the composer are business cases, policies and background knowledge.

Business cases represent the actual composition problem that consist of at least two types of objects – provided or initial data objects and required or goal objects. Policies guide the creation of models using rules, constraints and preferences. The major policy for the 09 challenge can be described as: Find a minimum number of services that solve the problem and arrange them in such a way that execution time is minimized using parallel ordering of services whenever possible. Finally, background knowledge stands for some kind of understanding of the “world”, i.e. the objects that populate this “world” and the relations between these objects. The composition challenge has a single OWL document with classes of objects, instances of those classes and subclass relationships between classes.

The output which is generated from the composition process depends on the specified use case, i.e. the purpose of the task and the kind of application that is intended to interpret the output. For the challenge 09 the interpreter is thought to be some kind of a BPEL engine, hence the output will be a BPEL process model.

3. Service Composition as Planning Problem

In past Web services composition challenges we have shown that Answer Set Programming (ASP) may be utilized in order to detect all possible solutions for a given composition problem. Having all solutions at hand, one can always figure out the best solution. However, this approach may fail whenever a very large amount of services is present due to the state space explosion problem. Such situations call for heuristic solvers that attempt to come up with a “good” solution which, as a trade-off, may not be the most favorable one. In order to be flexible in choosing a particular planning tool we transform the content of the registry (i.e. the service description), the background knowledge (i.e. the OWL document), and the problem (i.e. the challenge) to a standardized language supported by a number of tools.

3.1. PDDL

The planning domain description language PDDL has become a community standard for the representation and ex-
change of planning domain models. It gives the community the opportunity to exchange domain models and problems in a comfortable way and thus is a valuable instrument for the task of planning software evaluation and comparison.

PDDL is an action-centered language that uses pre- and post-conditions to describe the applicability and effects of actions. The syntax is inspired by Lisp. The description of the actions that characterize the behavior of the planning domain is separated from the description of the initial conditions, specific objects and goals that represent a particular problem instance. A planning problem is then created by the pairing of a domain description with a problem description. The PDDL specification is still evolving as new concepts (e.g. time, integer arithmetic, probabilistic effects) are added.

The following code snippet shows an example of a domain and problem definition. Obviously, the domain definition has some similarities with a service repository while the problem definition represents the business case (initial values and goal) as well as background knowledge with (simple) typed objects such as assigning types *doc* and *dir* to objects.

```plaintext
;; domain definition
(define (domain Jobs)
  (:requirements :typing)
  (:predicates (at ?x - doc ?y - dir))
  (:action CutPaste
   :parameters (?x - doc ?y - dir ?z - dir )
   :precondition (and (at ?x ?y))
   :effect (and (not(at ?x ?y))(at ?x ?z)))
)
;; problem definition
(define (problem p1) (:domain Jobs)
  (:objects a b c d e - doc
   x y - dir)
  (:init (at a x)(at b y)(at c x))
  (:goal (and(at a y)(at b x))))
```

Current versions of PDDL (2.1, 2.2 and higher) allow the specification of actions costs, soft and hard constraints as well as preferences [5]. This concepts may be used to apply policies to the planning process as described above. For instance, with action costs one could extend the previous example with metrics in order to guide the planning process as shown in the following code snippet:

```plaintext
(:functions (total-cost) - number)
...
  (:action CutPaste
   (increase (total-cost) 2))
...
  (:goal (and(at a y)(at b x)))
  (:metric minimize (total-cost))
```

However, while the standard is very comprehensive and evolving, there exists hardly any planner that has this standard implemented completely.

### 3.2. WSDL to PDDL

The Web service description language (WSDL) is an established industrial standard to describe the signature of operations (message types) and network and communication protocols (e.g. in/out, one way, SOAP). It is not always clear how such complex details can be represented in PDDL. Mapping WSDL to PDDL has been, for instance, demonstrated by [6]. However, both languages have only a few concepts in common and the author suggests an external annotation in order to support automatic translation and plan generation. Other approaches such as OWL-S [7] (S stands for Service) or the Web Service Modeling Ontology [8] (WSMO) basically use similar techniques. But, although these approaches have a number of advantages they all lack wider support from industry.

The basic mapping without additional annotation simply maps operations from WSDL to actions in PDDL. Preconditions and postconditions of PDDL-actions are derived from the WSDL input and output messages of operations.

### 3.3. Planning Algorithms

For the composition of complex business processes totally ordered plans are not very meaningful since they would lack completely the ability of concurrent execution of independent services. What is needed are algorithms that support this requirement as for instance the Graphplan algorithm [9]. Graphplan comes up with a solution (if one exists) that can be described as a set of sets of actions. The actions in one set are mutually independent and can be scheduled in a concurrent manner as the example below shows.

<table>
<thead>
<tr>
<th>Step</th>
<th>action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lecture A</td>
</tr>
<tr>
<td>2</td>
<td>Lecture A, Exam A</td>
</tr>
</tbody>
</table>

This kind of representation is strongly related to block-oriented languages and can be translated directly to an appropriate process language such as BPEL. Nevertheless, despite this similarity one has to be aware of the fact that it is possible that dependencies between actions (or operations when considering WSDL) may not be reflected correctly. For details on this issue see for instance [10], where the author compares block- and graph-oriented languages and their expressiveness. In order to overcome this potential problem a partial order planning algorithm (POP) [11] may be applied, but it needs some clarification whether the outcome of POP can be transformed always to BPEL.

### 4. Evaluation

We have evaluated our solution using the *blackbox* version 4.2 planning tool. The setting was Graphplan (which is the default algorithm), the computer had CPU with 3.40GHz and 1.00 Gigabyte memory.

2. The *blackbox* planner can be found at the website of Henry Kautz at www.cs.rochester.edu/~kautz/satplan/blackbox
The repository and the background knowledge has been translated to a PDDL domain using a very simple transformation procedure: Each operation from the WSDL document has been mapped to an action in the PDDL domain. The preconditions have been directly inferred from the WSDL document, while the effects have been expanded in order to reflect the background knowledge from the OWL document. Basically, when an action adds instances of a particular class to the state space it adds instances of more generic classes to the space, too. This reflects the fact that a certain type \textit{Type} can be substituted with a type that belongs to a subclass of \textit{Type}.

The results from several runs using the three Challenge Sets (from the past contest) are depicted in Table 1. The evaluation shows that mapping WSDL to PDDL and applying Graphplan to the transformation outcome gives results that are quite as good as the “best” solutions found by other teams. However, these results are preliminary and we think that there is room left for some performance improvements. Additionally, comparison to other planning tools has to be done in terms of performance and features of the tool such as support of constraints and preferences.

5. Conclusion and Outlook

The possible automatic composition of complex artifacts from basic building blocks is a major challenge to AI related algorithms and tools. MOVE serves as a generic platform that can be used to test and evaluate new approaches to this challenge. We have developed a number of components that help in the design of virtual organizations and inter-organizational business processes. In preparation for the Web services challenge 09 we have adapted our framework to this challenge and tested it successfully using problem instances from a past contest. However, further research is needed in order to improve the capabilities of our framework. In particular, we consider more complex business processes that include choices from the outside world and other sources of non-deterministic behavior and uncertainty as major research topics for the near future.

References


