An end-to-end configuration-based framework for automatic SWS composition

Patrick Albert
ILOG S.A
9, rue de Verdun
94253 Gentilly, France
palbert@ilog.fr

Laurent Henocque
LSIS Laboratory
Université de Saint-Jérôme
13397 MARSEILLE, France
laurent.henocque@lsis.org

Mathias Kleiner
LSIS Laboratory
Université de Saint-Jérôme
13397 MARSEILLE, France
mathias.kleiner@lsis.org

Abstract

Semantic Web Service (SWS) composition is a challenging AI problem. We describe a theoretical and experimental framework based upon finite model search for constrained object models to address this problem. In many AI situations the input is rather simple, and the results complex to obtain. SWS composition requests themselves can turn very complex, and the problem of building these requests can be viewed as an AI problem of its own. This paper presents an operational end to end approach to composing/publishing Semantic Web Services involving two main reasoning stages. Composing is first performed at the abstract level of goals (each roughly representing a discovery request), which yields a composition request at the workflow level. The resulting workflow is finally processed to generate a valid publishable semantic web service. We present experimental results obtained on industrial use cases during the DIP project.

1 Introduction

The automated composition of workflows or more recently of semantic web services is a very difficult AI problem attracting an intense activity, because of its applications to at least two wide areas: Business Process Modeling (BPM) and (Semantic) Web Services (SWS). Approaches to these problems are experimented using many formalisms and techniques, among which are Situation calculus [12], Golog [15], logic programming [22], type matching [6], Linear logic: [20], problem solving methods [8] and Ariadne [7], modal action logic [4], process algebra [17], AI Planning [19, 23, 18] (the list can’t be exhaustive) and constraint programming [1, 2, 11].

From a methodological standpoint, this work uses constraint programming, more precisely its generalization to configuration [13]. Configuration is a variant of finite model search for a kind of first order theories called constrained object models. We use and provide a Z [25] based formalization of these theories, thus ensuring independency from specific solving tools.

At a conceptual level, this work takes place within an emerging trend of research for semantic web composition, involving request languages (like ‘EaGLe’ [18]). The need for composition languages has emerged because of the complexity for an end user to formulate composition requirements that can be called ‘correct’ under the standard model theoretic assumption, i.e. when problem constraints are strong enough to forbid wrong solutions. Figure 1 illustrates in visual notation a composition request in our framework.

Figure 1. A composition request for the NMPC-bundle scenario

The main contribution of this paper is the description of a full fledged “end to end” composition framework covering all aspects from the user request down to the publication of the composite service. The scope of this research covers two independent languages, one to formulate composition goals, the other to denote SWS workflows, the implementation of the associated solvers, as well as an important algorithm for extracting publishable workflows from raw composition results. To the best of our knowledge no existing language offers the expressiveness covered by our approach, both from the request point of view or the generality of the computed composite services. Moreover, existing approaches require that the user formulates his requests as
full and valid instances of a complex language. This paper shows how to automatically generate full composition goals from simpler and incomplete requests. The reasoning aspects of SWS composition are achieved through a two stage approach, whereby goal-level composition results provide input to process-level composition as illustrated in Figure 2. This work extends the constraint based approach presented in [1, 2] where it was proved feasible to compose semantic web services orchestrations represented by workflows using configuration. The approach was experimentally validated in the DIP european project \(^1\). Due to space limitations, the present paper cannot enter into details of the composer but rather focuses on the main ideas in order to give a vision of the whole approach together with experimental results. Therefore we will not detail here the constrained object models nor the solving algorithms, but the reader can refer to [11] for complete information.

The plan of this paper is as follows. The current Section introduces the context (semantic web services, composition, configuration) and discusses the state-of-the-art. Section 2 describes the goal-level composition step. Section 3 briefly describes the integration with the process-level composition step as well as the automatic extraction of the composite service descriptions. Section 4 provides experiments through real-life scenarios and a comparison to existing work.

1.1 Brief introduction to Semantic Web Services

1.1.1 Semantic Web Services definition

Semantic Web Services (SWS) are commonly defined using both functional and behavioural specifications. There are currently two main formalisms for describing SWS: OWL-S historically, and more recently WSMO\(^2\) (our choice). WSMO separates the behavioural description (called an interface) into two different sections: the choreography describes the proper way for a client to consume its functionality, whereas the orchestration describes how the behaviour of a composed service is achieved using several clients (typically other SWSs). The functionality is described using a capability, basically in the form of the expected input and output message types.

1.1.2 Atomic Goals as SWS Abstractions

A user (or a composition tool) looking for a particular SWS will express requirements allowing its discovery. Such requirements are called (atomic) goals. Goal definitions closely relate to SWS capabilities, by specifying available inputs, expected outputs and constraints. In what follows, we call input and output roles the abstraction of input and output messages mentioned at the goal level.

---

\(^1\)http://dip.semanticweb.org

\(^2\)http://www.wsmo.org
1.2 Brief introduction to Web Service Composition

Composition is defined as the “act of combining and coordinating a set of Semantic Web Services” (SWS). Under such settings, “composition” naturally refers to the fact of computing an orchestration. This is what we call process-level composition. Several existing approaches focus solely on the conceptual level, creating an abstract specification of an orchestration at the level of goals. We will call this goal-level composition. Such an output is sometimes called an orchestration of goals in the literature.

1.3 Brief introduction to Configuration

Configuring means finding a set of related objects that obey the specification of an object oriented model. A configuration program is well described using a constrained object model (COM) in the form of a standard (UML) class diagram, together with well-formedness rules or constraints. Technically solving the associated enumeration problem can be done using various formalisms or technical approaches: extensions of the CSP paradigm [16, 27, 3, 21], knowledge based approaches [26], terminological logics [5, 14], logic programming (using forward or backward chaining, and non standard semantics) [24], object-oriented approaches[10]. All of them however amount to finite model search in first order logic.

1.4 Related Work

It is interesting to compare the expressive power and tools existing in different approaches to the same problem. Table 1 below summarizes this, with the following conventions: (+) = supported feature, (-) = not supported, (+/-) = partially supported. We consider the following features:

- (CompLevel) level of composition achieved i.e goal-level and process-level,
- (NAC) support for non-atomic choreographies i.e not only one-shot services,
- (WP) support for three sets of workflow patterns: (a) basic control flow (sequence, concurrency); (b) basic data flow (internal choice, external choice, synchronizing merge); (c) structural patterns (loops, multiple instances),
- (DR) support of data reasoning i.e reasoning on data ontologies besides strict equivalence of concepts,
- (SWS) support and integration with an existing SWS formalisms i.e DAML-S, OWL-S, WSMO, BPEL4WS,
- (Req) availability of a language for complex requests i.e able to express requirements other than input and output messages,
- (Disc) interaction with a discovery tool,
- (Tool) existence of a tool for automatic composition or at least specifications.

2 Goal-level composition

The introduction of a composition request language is motivated by several issues that arise in process-level composition. Briefly stated, the difficulties relate with:

- the formulation of complex composition requirements, as for instance is required at design time.
- the fact that a workflow composer aware of message types alone cannot infer the composite logic. Think for instance about the need of summing up costs, or subtracting a price to a budget.

2.1 A language for composition goals

We define a composition goal language (CG), at the abstract level of goals. The language offers:

- constraints on the non-functional properties (NFPs) of composed web services,
- constraints on the concepts carried by messages,
- constraints on the data flow. In particular symbolic links between messages, but also defining alternative paths (for instance, if I travel by car I want a parking in my hotel),
- constraints on the control flow for temporal requirements (for instance receive an order before I send a payment).

CG is defined as a COM hence allowing for assisted, semi-automatic or automatic composition of the request (i.e goal-level composition). Figure 3 is an overview of the COM using UML2 class diagrams. The CG language is also given a graphical representation through a concrete syntax based on UML2 activity diagrams (only the syntax) as illustrated in Figure 1 and Figure 4.

CG constrained object model example constraints

Due to space limitations we only present here a small excerpt from the full CG specification, which is available online3.

Apart from classical structure constraints that define valid

3http://www.lsis.org/config
Table 1. Comparing SWS Composition approaches

<table>
<thead>
<tr>
<th></th>
<th>CompLevel</th>
<th>NAC</th>
<th>WP</th>
<th>DR</th>
<th>SWS</th>
<th>Req.</th>
<th>Disc.</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal</td>
<td></td>
<td></td>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[15]</td>
<td>Golog</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>+</td>
</tr>
<tr>
<td>[8]</td>
<td>PSM</td>
<td>-</td>
<td>+</td>
<td>+/+</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>+</td>
</tr>
<tr>
<td>[7]</td>
<td>Ariadne</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>-</td>
</tr>
<tr>
<td>[4]</td>
<td>Modal</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td>-</td>
</tr>
<tr>
<td>[22]</td>
<td>LD</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td>[6]</td>
<td>Type Match</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+/+</td>
<td>+/</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[17]</td>
<td>Cashew</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>[23]</td>
<td>Shop2</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+/</td>
<td>+/</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[29]</td>
<td>GraphPlan</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-/-</td>
<td>-</td>
</tr>
<tr>
<td>[19]</td>
<td>Sword</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>-</td>
</tr>
<tr>
<td>[18]</td>
<td>STS</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>this paper</td>
<td>CBC</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>+/+</td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 3. CG COM excerpt overview

composition goals such as cardinality restrictions, the COM includes a number of constraints that support limited ontology reasoning. For instance, the configurator is able to infer that one concept is contained in another one, and may then add a mediator to the dataflow so that it extracts at runtime a required information.

- **IdentityFlow** expresses the semantic equivalence of two roles, hence later grounded to the same message in the composite workflow. **MergeFlow** and **DecisionFlow** allow for alternative paths to be (automatically) defined. **OperationFlow** and **AdaptationFlow** allow for transformations to be explicitly created beforehand.

- **Cardinality constraints** on **MergeFlow** and **DecisionFlow**

```
IdentityFlow : CGClass
MergeFlow : CGClass
DecisionFlow : CGClass
ActionFlow : CGClass
OperationFlow : CGClass
AdaptationFlow : CGClass

(IdentityFlow, MergeFlow, DecisionFlow, ActionFlow)
partition DataflowConstraint
(OperationFlow, AdaptationFlow)
partition ActionFlow
```
∀ n : DecisionFlow • #{n.sources} = 1
∀ n : MergeFlow • #{n.targets} = 1

- Sources and targets of IdentityFlow, DecisionFlow and MergeFlow must share the same concept:

∀ n : DataflowConstraint | 
(n ∈ IdentityFlow ∨ n ∈ MergeFlow ∨ n ∈ DecisionFlow) • 
sources(n) → concept = targets(n) → concept

2.2 Goal-level configurable requests: composition goals

Based upon the previous definition of the CG language, we see that a valid composition request is described by an instance of the CG language that satisfies all the constraints. The great innovation in using configuration is that a request to a composer can be formulated as a configuration request to the CG language. In other words, the user will be granted the right to formulate a partial statement, that the reasoning engine will complement to form a valid construct. This is illustrated by the gap that exists between Figure 1 (showing a composition request), and Figure 4 that is illustrates the composition goal that was automatically derived from it.

2.3 Integration with process-level composition

Once a fully valid composition goal is generated from the partially specified user request, and if no inconsistency is detected, a request to the workflow composer can be derived. This requires to first perform discovery to retrieve the choreography workflows from matching SWS. Since the previous phase may have strengthened the atomic goal constraints, the phase may yield fewer hits than could have been guessed from the start. Then, the composition goal can be translated to a partial workflow suitable for the next configuration stage by a set of straightforward, linear rules, as illustrated in Figure 5

3 Process-level composition

Process-level composition produces an orchestration from available SWS choreographies. We specify Choreographies and orchestrations using the same language, an adapted subset of the well known UML2 activity diagrams (UML2AD) called AD-S in the sequel. This language is based on Petri nets semantics. AD-S is also defined as a COM, allowing to use configuration for workflow composition as was proved feasible in [1, 2]. Figure 6 presents an overview of the AD-S metamodel. Figure 7, the resulting orchestration, illustrates the visual notation associated to this metamodel, and its expressiveness.

A trained UML designer will recognize two interruptible regions. Note that these elements were automatically generated during the publishing process, along with the extraction of the orchestration from the composite workflow.

3.1 Publishing

The last operation to perform is publishing the resulting composite SWS. We have defined a fix point simplification procedure that allows to generate from the results of
Figure 4. A computed composition goal for the NMPC-bundle scenario request

workflow composition a valid choreography and orchestration. This requires in particular to abstract from the participant choreographies workflow while preserving executability and conformance. The results produced by this final stage, illustrated in Figure 7, have been tested in a working environment producing on the fly composition and execution within the DIP project.

4 Experiments and analysis

We present experimental results for four scenarios. The scenarios have been integrated in an operational end-to-end SWS framework, from composition to execution. Configuration’s tasks are handled by Ilog’s tool JConfigurator. The producer-shipper-a and producer-shipper-bank scenarios are taken from [28]. In these simple cases, we do not make use of composition goals, and hence perform a single phase workflow composition. In both scenarios, an aggregated order confirmation is required from the user and implies interleaving SWSs execution. These scenarios can serve as a comparison with the STS composer proposed in [18], which is one of the most efficient available composers.

The producer-shipper-b scenario is a refinement proposed by a DIP use-case partner (SAP) where the involved choreographies have a more complex behaviour. There, the choreography describes a communication with 8 request/response patterns between the shipper and the producer. The request here is a simple composition goal restricting the SWS library but without any value or flow constraints.

The NMPC-bundle scenario serving as illustrations in this paper, has been joint developed with another DIP use-case partner (British Telecom).

Table 2 lists experimental results. All experiments were conducted with a java implementation on a Pentium IV 2.8GHZ with 512MB of RAM. The second column gives the number of existing SWS whereas the third column gives the number of candidate SWSs returned by discovery on the basis of the required atomic goals. The composition time includes all the composition operations, from the linear translation of the composition goal, to the configuration of the composite workflow and finally the extraction of a conformant orchestration. No special heuristics have been used during configuration beyond JConfigurator’s default ones.

On simple scenarios like PS-a and PSBank, when the
The number of available SWS is restricted to the necessary ones, the composer is efficient. It can be observed that increasing the number of available services (added services are carbon copies of the initial set) increases composition times above what we may expect for an “on-the-fly” composition. Note however that the search does not exploit any symmetry breaking technique.

The impact of using composition goals on scenarios PS-b and NMPC is obvious: thanks to discovery interaction, the number of candidate SWS is reduced to a set of the only services that are potentially useful, and the configuration search space is thus reduced. If we increase the total number of available SWS with services which do not match the atomic goals, the performance of composition indeed remains unaffected.

However the PS-b use case shows that with complex choreographies, increasing the number of services (again, we provided copies) has a measurable effect on composition times. This is due to the combinatorial explosion in the presence of an important number of nodes and edges.

The NMPC-bundle scenario illustrates the advantages of complex composition goals. Here we provided many services selling a network connection, but only one of them offers ADSL connection. Thanks to the propagation of UnaryValueConstraints (such as the NetworkDetails’ connectionType) to an atomic goal’s roles, we reduce the search space for the composer. Indeed, this restricts the list of candidate SWSs (in the current example only SWS offering ADSL connections will be discovered).

Finally, the ability to (manually, semi-automatically or automatically) fine-tune the composition request very precisely introduces a number of added workflow constraints. These additional constraints reduce the configuration space and discard unwanted orchestrations.

### 4.0.1 Comparison with existing composers

If we consider efficiency in terms of computation times, the comparison is difficult since the number of published experimental results on concrete reusable scenarios remains small. However we can compare two of our scenarios with the results obtained with the STS approach [18]. This is presented in Table 3. Although we found that the configuration-based composer suffers from a combinatorial explosion, it obviously stands the comparison with one of the most efficient known approaches, while providing additional features and expressiveness.

### 5 Conclusion

We have presented a correct and complete constraint based configuration framework that addresses the problem...

---

**Figure 7. A computed orchestration for the NMPC-bundle scenario composition goal (bold edges are object flows)**

**Table 3. SWS composition experimental comparison, times in seconds**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>STS time</th>
<th>Config time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS-a</td>
<td>9.4</td>
<td>1.33</td>
</tr>
<tr>
<td>PSBank</td>
<td>75.0</td>
<td>1.68</td>
</tr>
</tbody>
</table>
of SWS composition. The results on concrete scenarios are promising as well as the approach that treats this kind of composition problem using finite model enumerative search techniques. Among the benefits, the knowledge representation is close to existing SWS formalisms (description logics for data and workflows for services behaviour), solving is done using well-known constraint programming techniques, and the resulting solution is directly exploitable. Ongoing research aims at extending the AD-S language to support more workflow patterns, like loops, as well as extending the ontology reasoning potential of the metamodel.

References