Goal-oriented SWS composition prototype
DIP Deliverable 4.22

Patrick Albert
Christian de Sainte Marie
Laurent Henocque
Mathias Kleiner

January 5, 2007
SUMMARY

1. Summary
   This deliverable and its annexed documents and archives detail the principles, assumptions, organization of a prototype for a tool achieving automated goal directed composition of Semantic Web Services. The document and its annexes are intended to allow project partners to run the prototypes as a means of demoing or testing.

2. Contribution
   This work is an original attempt to use constraint based finite model search to achieve SWS composition, in the objective of both:
   
   - being usable as a helper application within the DIP tools that allow for manually or automatically editing Semantical Web Services,
   - reaching the highest possible level of functionality.

3. Interactions within DIP
   This deliverable is impacted by the WSMO specification for choreography and orchestration. The programming interface of the prototype is an input to DIP WP4 applications, as required for their integration. The prototype depends upon several DIP/WSMO APIs required to read/parse the specification of SWS. The prototype depends upon DIP discovery API required to obtain candidate SWS identifiers.

4. Target audience
   The deliverable is of particular interest to DIP participants involved in Choreography, Orchestration, Discovery, Process Mediation, Data Mediation, Use Cases.

5. Disclaimer
   The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
**Document Information**

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP6 – 507483</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Title</td>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://dip.semanticweb.org/">http://dip.semanticweb.org/</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document URL</td>
<td>Werner Janusch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DIP Deliverable 4.22**

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Goal-oriented SWS composition prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Package</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Service Usage</td>
</tr>
</tbody>
</table>

**Date of Delivery**

<table>
<thead>
<tr>
<th>Contractual</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>M36</td>
<td>30-Dec-06</td>
</tr>
</tbody>
</table>

**Status**

<table>
<thead>
<tr>
<th>version 0.1</th>
<th>final</th>
</tr>
</thead>
</table>

**Nature**

<table>
<thead>
<tr>
<th>prototype</th>
<th>report</th>
<th>dissemination</th>
<th>ontology</th>
</tr>
</thead>
</table>

**Dissemination Level**

<table>
<thead>
<tr>
<th>public</th>
<th>consortium</th>
</tr>
</thead>
</table>

**Authors (Partner)**

| see title page |

**Resp. Author**

<table>
<thead>
<tr>
<th>Patrick Albert</th>
<th>E-mail</th>
<th><a href="mailto:palbert@ilog.fr">palbert@ilog.fr</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>ILOG</td>
<td>Phone +33 (1) 49-08-35-00</td>
</tr>
</tbody>
</table>

**Abstract**

(For dissemination)

This deliverable implements the specification detailed in deliverable D4.12

**Keywords**

composition, configuration, constraint, policy, finite model, search, semantic web service, web service, workflow, modeling

**Version Log**

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>17-Nov-06</td>
<td>1</td>
<td>Mathias Kleiner</td>
<td>Created</td>
</tr>
<tr>
<td>7-Dec-06</td>
<td>2</td>
<td>Mathias Kleiner</td>
<td>Initial content completed</td>
</tr>
<tr>
<td>12-Dec-06</td>
<td>3</td>
<td>Mathias Kleiner</td>
<td>Refined licencing and third party tools info</td>
</tr>
<tr>
<td>23-Dec-06</td>
<td>4</td>
<td>Mathias Kleiner</td>
<td>Reviewers comments modifications</td>
</tr>
</tbody>
</table>

**Reviewers**

<table>
<thead>
<tr>
<th>Barry Norton</th>
<th>E-mail</th>
<th><a href="mailto:b.j.norton@open.ac.uk">b.j.norton@open.ac.uk</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>Open University</td>
<td>Phone +44 (0) 1908 659399</td>
</tr>
<tr>
<td>Alex Simov</td>
<td>E-mail</td>
<td><a href="mailto:alex.simov@ontotext.com">alex.simov@ontotext.com</a></td>
</tr>
<tr>
<td>Partner</td>
<td>Ontotext</td>
<td>Phone (+359 2) 9768 310</td>
</tr>
</tbody>
</table>
# Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Galway | NUIG | Dr. Sigurd Harand  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: sigurd.harand@deri.org  
Tel: +353 91 495112 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmtnez@bankinter.es  
Tel: 916234238 |
| British Telecommunications Plc. | BT | Dr. John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
Email: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowlatt,  
Essex County Council,  
PO Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
E-mail: maryr@essexcc.gov.uk  
Tel: +41 (0)1245 436524 |
| Forschungszentrum Informatik | FZI | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | UIBK | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Logo</th>
<th>Contact Person</th>
<th>Address/Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOG SA</td>
<td>ILOG</td>
<td>Christian de Sainte Marie</td>
<td>9 Rue de Verdun, 94253, Gentilly, France</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a></td>
<td>Tel: +33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td>inubit</td>
<td>Torsten Schmale, inubit AG</td>
<td>Lützowstraße 105-106 D-10785 Berlin, Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:ts@inubit.com">ts@inubit.com</a></td>
<td>Tel: +49 30726112 0</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>iSOCO</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D</td>
<td>Pedro de Valdivia 10 28006 Madrid, Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a></td>
<td>Tel: +34 913 349 797</td>
</tr>
<tr>
<td>MDR Partners</td>
<td>MDR</td>
<td>Rob Davies</td>
<td>MDR Partners 8 St. Andrew Street, Hertford, Herts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United Kingdom, SG14 1JA</td>
<td>E-mail: <a href="mailto:rob.davies@mdrpartners.com">rob.davies@mdrpartners.com</a></td>
</tr>
<tr>
<td>Hanival Internet Services GmbH</td>
<td>HANIVAL</td>
<td>Alexander Wahler</td>
<td>NIWA WEB Solutions Hanival Internet Services GmbH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:wahler@niwa.at">wahler@niwa.at</a></td>
<td>Kirchengasse 13/1a A-1070 Wien</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +43 131 95843 11</td>
<td></td>
</tr>
<tr>
<td>The Open University</td>
<td>OU</td>
<td>Dr. John Domingue</td>
<td>Knowledge Media Institute, The Open University, Walton Hall</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a></td>
<td>Milton Keynes, MK7 6AA, UK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +44 1908 655014</td>
<td></td>
</tr>
<tr>
<td>SAP AG</td>
<td>SAP</td>
<td>Dr. Elmar Dorner</td>
<td>SAP Research, CEC Karlsruhe SAP AG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a></td>
<td>Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +49 721 6902 31</td>
<td></td>
</tr>
<tr>
<td>Sirma AI Ltd.</td>
<td>Sirma</td>
<td>Atanas Kiryakov, Sirma</td>
<td>Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a></td>
<td>135 Tzarigradsko Chausse, Sofia 1784, Bulgaria</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +359 2 9768 303</td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td>Address</td>
<td>Contact Information</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------</td>
<td>----------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Unicorn Solutions Ltd, Malcha Technology Park 1, Jerusalem 96951, Israel</td>
<td>E-mail: <a href="mailto:Jeff.Eisenberg@unicorn.com">Jeff.Eisenberg@unicorn.com</a> Tel.: +972 2 6491111</td>
<td></td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Vrije Universiteit Brussel Pleinlaan 2, G-10, 1050 Brussel, Belgium</td>
<td>E-mail: <a href="mailto:Pieter.De.Leeheer@vub.ac.be">Pieter.De.Leeheer@vub.ac.be</a> Tel.: +32 (0) 2 629 3749</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

1 **Introduction**  
1.1 Scope ................................. 1  
1.2 Context for goal oriented SWS composition ..................................... 1  
1.3 Composing as a Finite Model Search problem .................................... 2  
1.4 A workflow language for choreographies and orchestrations ...................... 3  
1.5 A language for abstract composition goals ......................................... 3  
1.6 Brief introduction to configuration .................................................. 4  
1.7 Related work ..................................................................................... 4  

2 **Fact Sheet**  
2.1 Deliverable name .............................................................................. 5  
2.2 Contact person with contact details .................................................. 5  
2.3 Brief description of purpose ............................................................. 5  
2.4 Brief description of scope ................................................................... 5  
2.5 Brief description of functionality ....................................................... 5  
2.6 Type of API (e.g. JAVA, WSDL) .......................................................... 6  
2.7 Format of program input ................................................................. 6  
2.7.1 Abstract model for composition goals ............................................. 7  
2.7.2 Graphical representation .............................................................. 9  
2.8 Format of program results ............................................................... 10  
2.9 Files attached to the current document ............................................. 10  

3 **Detailed License Information** ............................................................. 11  
3.1 License .............................................................................................. 11  
3.2 How to obtain a license ...................................................................... 11  

4 **Detailed Third Party Tool Information** ................................................. 12  
4.1 JAVA SDK 1.5.0+ ........................................................................... 12  
4.2 ILOG JCONFIGURATOR ................................................................. 12  
4.3 WSMO Studio .................................................................................. 13  
4.4 Visual Paradigm for UML ................................................................. 13  

5 **Technical Requirements for Using the Prototype** .................................... 14  
5.1 JAVA SDK 1.5.0+ ........................................................................... 14  
5.2 ILOG JCONFIGURATOR ................................................................. 14  

6 **Detailed Information on How to Use/Evaluate the Prototype** ....................... 15  
6.1 Download ........................................................................................ 15  
6.2 Installation ....................................................................................... 15  

7 **Known Issues and Limitations** ............................................................. 16  

8 **Roadmap on Future Plans for Further Developing the Prototype** .................. 17  

9 **User Manual** .................................................................................. 18  

10 **Reference Manual** .......................................................................... 19
1 INTRODUCTION

This deliverable presents a prototype workflow composer for DIP based upon constraint programming technologies. The present document is expected to provide enough information to operate the prototype without any help. As a project partner, ILOG is ready and willing to help other partners in adapting the program to their needs. The rest of this section briefly presents the scope and context of configuration based workflow composition, and can be skipped by a reader already aware of these issues.

1.1 Scope

We place ourselves in the scope of automatic or computer aided SWS composition. The basic assumptions for composing SWS is that there exists a repository of SWS that are potential candidates for composition and can be accessed via discovery, as well as a directory listing of mediators needed to adapt messages between SWS having incompatible message ontologies.

We also assume that the composition process is goal oriented: the user must formulate a request to the composer in the form of a “composition goal”: a loose or precise abstract statement of the constraints posted on elementary goals that must participate to a composition. The requirement for an abstract composition language for the composition of semantic web service$^1$ has been acknowledged by the DIP Specification Deliverable 4.12$[8]$, the main input for the current work. The D4.12 also precisely defines the scope of the current prototype and the reader is kindly directed to this source for further details. In order to be sufficiently self contained however, the present introduction borrows some material from the deliverable D4.12 and from the articles $[1]$ and $[2]$. A reader aware of configuration techniques and of configuration applied to web service composition may skip this part or read it quickly. The DIP deliverable 4.10 proposes a formal presentation of composite goals that bears some relations with the language defined in D4.12.

The current deliverable also follows from an earlier DIP prototype delivered in June 2006 under the id D4.15.

1.2 Context for goal oriented SWS composition

Our approach to goal oriented SWS composition can be sketched by saying that the (workflows that implement the) choreographies of the SWS matching atomic goals are combined to form a valid orchestration for a resulting composite SWS under the constraints set by a request to the composer called a composition goal. The process at a glance is the following:

- the user defines his request to the composer in the form of a composition goal involving atomic goals plus constraints

$^1$There are many justifications for the statement of complex composition requests. An example: a composite SWS needs to sum up cost data output by several participants: there is no way for an automated process to determine the need for such a summation from the choreographies alone.
• atomic goals present in the request are used to discover matching semantic web services,

• the choreographies of matching SWS are input to the composer as elementary bricks of the solution,

• the composition goal is input to the composer as constraints that restrict the possible constructions,

• the composer calls a configurator to produce a valid orchestration from the available choreographies under the specified constraints,

• the result is then exported as a SWS description involving a valid orchestration description, in the WSML-AD extension

1.3 Composing as a Finite Model Search problem

A large body of research addresses the SWS/Workflow composition problem as a reasoning task in a first order logic variant. We chose to consider this problem using an approach that falls within the field of finite model search. The underlying logic has the expressive power of description logics, and allows the statement of constrained object models.

The specification of the constrained object models is achieved via a combination of UML2 Class diagrams for the visual part, and the Z specification language for the formal specification of model constraints. We chose not to use the OCL2 language for this purpose. Arguments supporting this choice as well as complete examples can be found in [5, 8].

Finite model search for such theories is performed using a constraint based configurator call ILOG JConfigurator. JConfigurator accepts as input a constraint based variant of description logics[16], in which the problem specification is translated (this step is still manual but could be automatized in the future).

Configuration emerges as an AI technique with applications in many different areas, where the problem can be formulated as the production of a finite instance of an object model subject to constraints. Reasoning about workflows falls into this category, because a workflow description is an instance of a given metamodel (as is the UML metamodel for activity diagrams [15]). Composing workflows can be seen as a configuration problem: indeed, it requires to introduce an arbitrary number of previously non existing transitions (fork, join, split, merge, transformations, pre-defined user-interactions sequences, etc), and to interconnect input and output message pins which have compatible types.

The user of a workflow composition system expects in return for his input a complete “composite” workflow, that interleaves the execution of several of the elementary argument workflows, while ensuring that all possible integrity constraints as well as the constraints formulated in the composition goal request remain valid. Among integrity constraints are those that stem from the metamodel itself: for instance some constraints state that two or more workflows should not enter deadlock situations, all waiting for some other to send a message. Other constraints are more problem specific and mentioned in the request, like those stating for instance that an item being shipped is indeed the one that was produced.
1.4 A workflow language for choreographies and orchestrations

We consider SWS having choreographies defined using a close variant to a subset of UML2 activity diagrams [15]. Such a definition is not mandatory for a SWS to be correctly published within DIP, but is required for composition. The rationale is that since the composer operates at the structure level of choreographies and orchestrations, this structure must be made explicit. The execution model of WSMX based on ASMs does not warrant the possibility of retrieving an implicit structure from the ASM specification.

The visual workflow language chosen for choreographies and orchestrations was formally specified as part of the DIP D3.4, D3.5 and D3.9 deliverables in their common DIO annex [6]. The underlying operational semantics is that of colored Petri nets, where messages (tokens) have types. The composition process however doesn’t need to consider the operational semantics of the workflow language, but rather focus on the properties of the corresponding metamodel. In other words, the composer deals with the static structure of workflows alone and the message ontologies, but does not need to emulate the message flows.

More precisely, we treat SWS composition as the process of connecting input and output message flows to preexisting or newly introduced workflow elements, as are for instance fork, join, decision, merge nodes, mediators or auxiliary user input/output handling actions. Hence the only elements retained for composition are the structural properties of argument workflows, messages, and transformations. We do not need to emulate workflows in any sense, but can however formulate constraints that to some extent guarantee the viability of the result [2].

This general approach to workflow composition originates from several research communications [29, 11, 21] and was experimented using configuration techniques in [1]. We have illustrated in the D4.12 [8] our central assumptions by considering the rather complex Producer/Shipper composition problem introduced in [21]. The problem is to compose a valid workflow from a producer workflow and a shipper workflow. One difficulty of this use case is that the execution of both workflows must be interleaved. The producer outputs results that must be fed into the shipper so that both “offers” can be aggregated and presented to the user. This inter-connection remains unknown to the user. The reader is kindly directed to [8, 6, 7, 5] for details.

1.5 A language for abstract composition goals

Composition requests abstract the entire set of valid compositions that can be produced from elementary SWS matching their atomic goals and cannot be expressed using a workflow language as UML-AD for instance. In the general case, more complex interrelation may occur, for instance to state the some data must represent the sum of several other messages, or that some messages must be aggregated to build a message from a more complex ontology (for instance aggregating three integers to build a date. The composition goal language proposed in this deliverable supports such definitions.

\footnote{For instance some constraints prevent simple deadlock situations that arise when two threads wait for a message from the other}
1.6 Brief introduction to configuration

A configuration task consists in building (a simulation of) a *complex product* from *components* picked from a catalog of *types*. Neither the number nor the actual types of the required components are known beforehand. Components are subject to *relations*, and their types are subject to *inheritance*. *Constraints* (also called well-formedness rules) generically define all the valid products. A configurator expects as input a fragment of a target object structure, and expands it to a solution of the configuration problem, if any. This problem is semi-decidable in the general case.

A configuration program is well described using a *constrained object model* in the form of a standard class diagram, together with well-formedness rules or constraints. Technically solving the associated enumeration problem can be made using various formalisms or technical approaches: extensions of the CSP paradigm [19, 13], knowledge based approaches [26], terminological logics [20], logic programming (using forward or backward chaining, and non standard semantics) [25], object-oriented approaches [17, 26]. Our experiments were conducted using the object-oriented configurator Ilog JConfigurator [17].

1.7 Related work

Automated workflow composition is a field of intense activity, with applications to at least two wide areas: Business Process Modeling and (Semantic) Web Services. Tentative techniques to address this problem are experimented using many formalisms and techniques, among which Situation calculus [18], Logic programming [23], Type matching: [10], Coloured Petri nets: [29, 11], Linear logic: [22], Process solving methods [3, 14, 27], AI Planning [4], Hierarchical Task Network (HTN) planning [24, 28], Markov decision processes [12].
2 Fact Sheet

2.1 Deliverable name

The prototype SWS composer is called GOBAC (for GOal BAseD Composer).

2.2 Contact person with contact details

For technical information about GOBAC or questions, please contact Mathias Kleiner, by email at mkleiner@ilog.fr, or by post at

ILOG
9 rue de Verdun BP85
94253 Gentilly France

2.3 Brief description of purpose

GOBAC provides a helper application for design time composition of SWS.

2.4 Brief description of scope

GOBAC addresses the problem of Semantic Web Composition for DIP. GOBAC exploits as a metamodel for workflows a large subset of UML2 activity diagram concepts, hence addresses the problem in a true generality.

GOBAC does not currently address scalability issues. Computation times are impacted by many factors, including the presence/absence of redundant constraints that help filter out unwanted configurations.

2.5 Brief description of functionality

The GOBAC prototype is packaged as a WSMO Studio plugin, and thus allows to use a common environment with other WSMO Studio tools.

The program also provides an API in order to integrate it in a Java program, this API is further documented in Section 10.

GOBAC allows one to generate a valid orchestration corresponding to a composition goal request, that can be emulated using one of the DIP implementation platforms, i.e. WSMX or IRSIII. The current state of the prototypes invokes the DIP discovery tool.

The program expects as input:

- a composition goal: the main composer request, as an XMI 2.1 file.
- the choice of a WSMO repository where candidate web services descriptions are stored.
Given such an input, GOBAC produces as its output an orchestration obeying the constraints mentioned in the composition goal. The result can be interpreted as an UML2 activity diagram, naturally, or as the orchestration of the resulting composite service, and is naturally exported as a WSML-AD description (see [9] for a specification of this extension).

2.6 Type of API (e.g. JAVA, WSDL)

GOBAC is distributed as a WSMO Studio plugin, which requires the following third party libraries:

- JConfigurator 2.1 libraries

We also provide a JAVA API for tools integration.

GOBAC is built using the JAVA SDK 1.5.0+.

2.7 Format of program input

Composition goals are currently described using a specific format, not yet considered for a WSML extension. The prototype accepts the specification of its input in the form of an XMI 2.1 file produced by an UML2 compliant visual editor.

The specific composition goal language may become an extension to WSML as currently are the Activity Diagram and Cashew based choreography/orchestrations to WSML. Obviously, composition goal language might as well be implemented in the form of a WSMO ontology. Using such an ontology could allow to store and retrieve composition goals from a repository. Our recommendation is to let the ontology sketched in deliverable 3.10 to fully account for the D4.12 composition goal language.

The state of the prototype here documented requires the composition goal to be stated in a XMI file, exported by a compliant visual editor. The visual editing uses a specific adaptation of activity diagrams based on mandatory stereotypes and tagged values. We provide a full description of how to describe a composition goal, as well as a template file for those stereotypes usable in a visual editor called Visual Paradigm for UML.

Choreography related input could be read from DIP SWS descriptions using the WSML parser, to obtain machine processable data structures. Our activity diagram language is available as a WSML extension, described in D3.8 [9] on choreography and orchestrations. However, there is not yet a stable API that could be used to read those choreographies descriptions. As a consequence, the composer currently reads the choreography from an XMI file exported through any UML2 compliant visual editor. This file must be referenced in the WS non functional property with key value http://www.massidia.fr/semanticweb/XMIChoreography.
2.7.1 Abstract model for composition goals

We present in Figures 2.1 and 2.2 the composition goal language used to formulate requests to the composer, as an UML abstract model together with Z constraints specifying well-formed composition goals.

![Diagram of the abstract model for composition goals](image)

**Figure 2.1: Goals, roles and concepts**

**Semantics**

- **Atomic goals**: abstractions of SWS’s. Used to perform discovery thus making the matching SWS’s available for composition in the solution workflow.
- **Roles**: Inputs and outputs of matching SWS’s. Internal roles can be used to denote intermediate objects in the orchestration. Each role has a concept taken from an ontology.
- **Value Constraints**: the solution workflow respects any given value constraints.
  - **Unary Value Constraints**: the solution workflow ensures that the specified object will respect given (ranged) values.
  - **Relational Value Constraints**: the solution workflow ensures that the specified objects will respect given constraint between them.
- **Dataflow Constraints**: the solution workflow ensures the existence of a specific dataflow between sources and targets:
Goals and Roles constraints

- **IdentityFlow**: the dataflow between the source and the targets makes no change to data. Source and targets concepts must be the same.
- **OperationFlow**: the dataflow will perform an operation on sources tokens values to obtain targets. Only applicable to integers and floats.
- **MediationFlow**: the dataflow between sources and targets needs to use a specific mediator.
- **AggregationFlow**: the dataflow will aggregate sources into the composite concept target.
- **ExtractionFlow**: the dataflow will extract parts of the composite concept source into the targets.
- **DecisionFlow**: the dataflow will go from source to different targets depending on the constraint text. Source and targets concepts must be the same.
- **MergeFlow**: the dataflow takes any incoming source and delivers it to target. Source and targets concepts must be the same.

Z Constraints

- Goals to Roles relations reciprocity:

  \[
  \forall n : \text{AbstractGoal} \bullet \\
  \text{inputs}(n) = \{e : \text{Role} \mid e\.goal = n\} \\
  \forall n : \text{AbstractGoal} \bullet \\
  \text{outputs}(n) = \{e : \text{Role} \mid e\.goal = n\} \\
  \forall n : \text{AbstractGoal} \bullet \\
  \text{internals}(n) = \{e : \text{Role} \mid e\.goal = n\}
  \]
• Sources of dataflows must be of class OutputRole or InternalRole:
  \[\forall n : \text{Role} \mid \#(n.\text{isSourceOf}) > 1 \bullet \]
  \[n \subseteq \text{OutputRole} \lor\]
  \[n \subseteq \text{InternalRole}\]

• Targets of dataflows must be of class InputRole or InternalRole:
  \[\forall n : \text{Role} \mid \#(n.\text{isTargetOf}) > 1 \bullet \]
  \[n \subseteq \text{InputRole} \lor\]
  \[n \subseteq \text{InternalRole}\]

• Sources and targets of IdentityFlow, DecisionFlow and MergeFlow must share
  same concept:
  \[\forall n : \text{DataflowConstraint} \mid \]
  \[n \subseteq \text{IdentityFlow} \lor n \subseteq \text{MergeFlow} \lor n \subseteq \text{DecisionFlow} \bullet\]
  \[\text{sources}(n).\text{concept} = \text{targets}(n).\text{concept}\]

• DecisionFlow has only one source:
  \[\forall n : \text{DecisionFlow} \bullet \]
  \[\#(n.\text{sources}) = 1\]

• MergeFlow has only one target:
  \[\forall n : \text{MergeFlow} \bullet \]
  \[\#(n.\text{targets}) = 1\]

• ExtractionFlow has only one source, which concept is composite:
  \[\forall n : \text{ExtractionFlow} \bullet \]
  \[\#(n.\text{sources}) = 1\]

• AggregationFlow has only one target, which concept is composite:
  \[\forall n : \text{AggregationFlow} \bullet \]
  \[\#(n.\text{targets}) = 1\]

2.7.2 Graphical representation

We provide a graphical representation based on UML2 Activity diagrams in order
to draw composition goals in a user-friendly environment, using tools such as “Visual
Paradigm for UML” or “Poseidon”. Figure 2.3 lists all constructs, stereotypes and their
graphical notation. Concepts for roles are defined using text tagged value “concept”,
and goals URI are defined using the tagged value “URI”.

It may be noted that the aforementioned elements are optional. It is not required
in the general case for instance to attach ontological information to all of the roles,
since this information is redundant with WSML, unless this is required for discovery.
2.8 Format of program results

GOBAC produces an orchestration that can be published to DIP using the WSML-AD extension. The orchestration can then be processed to generate a valid orchestration in the context of either DIP implementation platform (WSMX/IRSIII).

The use of the WSMX platform for execution requires to generate the set of ASM rules that implement the choreography and orchestration. The specification of token flow semantics in the state of the art WSMX ASMs is not currently available (although under study), which postpones this possibility.

The use of the IRS-III platform as a service repository and execution platform requires that the choreographies be translated from Cashew to Activity Diagrams, which is detailed in D3.8, and that the orchestration produced be translated back to Cashew for execution, which is detailed in D3.9.

2.9 Files attached to the current document

The DIP Deliverable 4.22 consists in the present document plus the following files:

1. composition_GOBAC.zip: a zip archive of the WSMO Studio composer plugin.
2. VP_template.zip: a zip archive of a Visual Paradigm for UML template file for building composition goals with appropriate stereotypes.
3. GOBAC_examples.zip: a zip archive containing the necessary XMI files for two examples, as well as images of the diagrams and requests.
3 Detailed license information

This section describes the license and conditions granted by ILOG to test or use the prototype within the DIP project.

3.1 License

This deliverable has the dissemination level status.
ILOG grants a limited number of free licenses or the underlying ILOG products necessary to the testing of GOBAC by the relevant project’s partners within the project timeframe.

3.2 How to obtain a license

For any licensing information regarding the ILOG JConfigurator commercial product embedded in the current prototype, please contact Christian de Sainte Marie at ILOG:

ILOG
9 rue de Verdun BP 85
94253 Gentilly France
4 Detailed third party tool information

4.1 JAVA SDK 1.5.0+

LICENSE
The Sun license for the Java SDK 1.5.0+ is available at:

http://java.sun.com/j2se/1.5.0/jdk-1_5_0_04-license.txt

Third party licenses available at:

http://java.sun.com/j2se/1.5.0/j2se-1_5_0-thirdpartyreadme.txt

OBTAIN A LICENSE
The license is granted by accepting the proposed conditions at download time.

DOWNLOAD
The Windows installer is available for download at the url:

http://java.sun.com/j2se/1.5.0/download.jsp

INSTALLATION
Simply run the installer.

4.2 ILOG JCONFIGURATOR

LICENSE
This is a commercial product, sold or distributed under the terms of a commercial licence.

OBTAIN A LICENSE
The product, and license, may be obtained from ILOG, via Christian de Sainte Marie, or Patrick Albert, who can be reached at the phone number +33149083500, or by mail at csma@ilog.fr, palbert@ilog.fr, or by post at

ILOG
9 rue de Verdun BP85
94253 Gentilly France

DOWNLOAD
No download available.

1 Accessed in December 2006
2 Accessed in December 2006
3 Accessed in December 2006
INSTALLATION
Simply execute the CD rom, that should start automatically then follow the instructions. Commercial licenses entitle to full support.

4.3 WSMO Studio

LICENSE
WSMO Studio is under GNU lesser general public license. See http://www.wsmostudio.org/license2.html for details.

DOWNLOAD
WSMO Studio can be downloaded at http://www.wsmostudio.org/download.html.

4.4 Visual Paradigm for UML

LICENSE AND DOWNLOAD
The tool and license information is available at http://www.visual-paradigm.com. Visual paradigm proposes a free community license. The license allowing for XMI generation however is commercial.
5 TECHNICAL REQUIREMENTS FOR USING THE PROTOTYPE

The prototype is available for platforms running Windows XP, and requires at least 512MB of RAM. The program performance is satisfactory on machines having a 2 Giga-Hertz processor or faster. The machine must have at least 1 GB of free disk space if all components are to be installed (Java SDK inclusive).

5.1 JAVA SDK 1.5.0+

Running the prototype or the demo requires having installed a release of Java 1.5.0++.

5.2 ILOG JCONFIGURATOR

Running the prototype or the demo requires having installed a release of ILOG JConfigurator 2.1, under the patch level 7. Please contact Christian de Sainte Marie for any question regarding this product.
6 Detailed information on how to use/evaluate the prototype

6.1 Download

The prototype archives can be downloaded from http://www.massidia.fr/semanticweb/gobac/

6.2 Installation

The steps for installing the prototype are:

1. Install the J2SE 1.5.0 SDK (or higher) on your computer in a path containing NO SPACES.

2. Set your JAVA_HOME environment variable to your java sdk directory. (Under Windows : My Computer ⇒ properties ⇒ advanced ⇒ environment variables ⇒ system variables ⇒ new ⇒ JAVA_HOME)

3. Install JConfigurator 2.1

4. Extract the GOBAC_composition.zip archive in your WSMO Studio plugins directory.

5. Copy the following libraries from the lib directory of your JConfigurator installation path to the libs directory of the composer plugin: log4j.jar, crimson.jar, jconfigall.jar

6. The plugin cannot directly use the discovery plugin as it does not provide any exported packages. Hence, the discovery libraries are directly included in the prototype release. However, licensing does not allow to include the kaon2 necessary library for discovery in the release. Copy the kaon2 library (kaon2.jar from kaon2.semanticweb.org) into the libs directory of the composer plugin.

You can now use the prototype inside WSMO Studio, by displaying the Composer view.

Further details regarding installation issues are provided in the deliverable archives.
7 KNOWN ISSUES AND LIMITATIONS

- The composer does not work at the instance level of messages. As a consequence, composition goal’s value constraints have no effect on the orchestration.

- Consecutive calls to discovery might give different results although the request is exactly the same. A possible workaround is to close and reopen the studio, but this may not always work. Newer versions of the discovery libraries may fix this issue.

- Discovery does not support preconditions in queries, therefore the composer plugin ignores them when computing the atomic goals.
8 **Roadmap on future plans for further developing the prototype**

The current prototype is the final prototype.
9 User Manual

The composer plugin usage only consists in 3 steps:
1) Select the working Studio repository
2) Import the XMI file specifying the composition goal
3) Click on “compose” to populate the orchestration.
4) A new view containing the results will appear. If a solution is found, you have the choice to save it as a WSML file in a project.

Figure 9.1: Importing the composition goal

Figure 9.2: Composition results
10 Reference Manual

The tool provides an entry point in order to integrate the composer into a java program. A Javadoc is included with the prototype archive. The main classes and methods exported by the library jcomposer.jar are:

```java
public class JComposer
public AD composeFromCG(CompositionGoal compositionGoal, ArrayList<AD> choreographies)
(Creates an orchestration)

public class AtomicGoal extends AbstractGoal
public class CompositionGoal extends AbstractGoal
public ArrayList<AtomicGoal> getAtomicGoals()

public class Imports
public static CompositionGoal parseXMIGC(String path)
(Creates a composition goal, path is the absolute path to a local file containing the composition goal in a XMI 2.1 format)

public static AD parseXMIAD(String path, CompositionGoal CG)
(Creates a choreography, path is the absolute path to a local file containing the choreography in a XMI 2.1 format)

public class Exports
public static void generateADWSML(AD orchestration, String path)
(Generates a WSML-AD orchestration file, path is the absolute path where the orchestration should be written)

public class Conversions
public static org.wsmo.service.Goal convertToWSMOGoal(AbstractGoal ag) throws InvalidModelException, ParserException
(Creates a WSMO Goal usable for discovery)
```

The JComposer class also provides attributes for tweaking the composer solving preferences. Those will be further documented in the deliverable archives, and ILOG is willing to provide its help to adapt the product to specific partner requirements.
REFERENCES


