WP3: Service Ontologies and Service Description

D3.10

Goal Description Ontology

Michael Stollberg (UIBK)
Martin Hepp (UIBK)

July 10, 2006
SUMMARY

This deliverable presents a conceptual and ontologized model for describing goals as client requests for Web service usage. Within the MOF model used in DIP / WSMO for defining description models, this is an M2 layer (meta-model layer) ontology. We refer to it as the Goal Description Ontology, short GDO.

The purpose of the concept of goals is to enable automated Web service usage for clients on the knowledge-level. Instead of formulating technical requests for using Web services that do not rely on any common structure, clients shall be enabled to formulate their objectives as goals that abstract from technical details and concomitantly carry all information that is necessary for automated discovery, composition, and execution of Web services. Therefore, goals are defined by declarative descriptions that present user objectives. Their structure follows a generic model that ensures that all information for automated resolution of the goal are available. This is what we refer to as a goal model.

The goal model presented here is intended to expand the current goal definition in WSMO. It integrates and extends the definition and usage of goals for automated Web service usage from respective DIP technologies, in particular the concept of goals in WSMO, the goal model for the DIP composition technology, and usage of goals for automated Web service invocation in WSMX and IRS. For elaboration, this report first exposes the motivation for goals as a central element of Semantic Web service architectures. Then, we examine existing approaches for goal-driven architectures from different AI disciplines, identify the requirements for defining goals for Web service usage, and expose the related DIP approaches. On basis of this, we define a metamodel for describing goals whose main aspects are:

1. differentiation of 3 goal types: Atomic Goals that allow specification client objectives in terms of requested functionalities and client interfaces as a means for automated Web service invocation; Composite Goals for specifying client objectives as desired workflows that can not be expressed as Atomic Goals, and Abstract Goals as the common part of Atomic and Composite Goals for specifying requested functionalities in an abstract manner

2. differentiation of Goal Templates and Goal Instances: the former denotes schematic goal definitions whereof the latter is created for expressing a concrete client objective via input instantiation; Goal Instances can only be created from Atomic or Composite Goals.

We provide formal, language independent definitions of the central concepts and properties of the goal model, and expose its adaption to WSMO as the framework underlying the DIP technology. With respect to the progress and status of the DIP project, the presented goal model will most likely not be fully realized within the dependent DIP technologies. Rather, it provides a model derived from experiences and insights gained throughout the project that is intended to serve as a basis and input for follow-up projects that will apply and extend the DIP technology.
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.
## Abstract
This deliverable presents the **Goal Description Ontology** short **GDO**, a meta-model and language for describing goals as client requests for Web service usage. The elaborated goal model is not intended to be completed applied within the DIP technology but rather presents a future-oriented research report as input and a basis for follow-up research and development efforts for Semantic Web services.

### Keywords
- Goals
- goal-driven architecture
- meta-layer description ontology

### Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>28-Jan-06</td>
<td>1</td>
<td>Michael Stollberg</td>
<td>Initial document</td>
</tr>
<tr>
<td>05-May-06</td>
<td>2</td>
<td>Michael Stollberg</td>
<td>First Completed Draft</td>
</tr>
<tr>
<td>12-June-06</td>
<td>3</td>
<td>Michael Stollberg</td>
<td>finished draft sent for DIP internal review</td>
</tr>
<tr>
<td>29-June-06</td>
<td>4</td>
<td>Michael Stollberg</td>
<td>final version, submitted</td>
</tr>
</tbody>
</table>

### Reviewers

- **Laurent Henocque**
  - **E-mail**: henocque@esil.univ-mrs.fr
  - **Phone**: +33 - 04 91 28 83 66

- **Barry Norton**
  - **E-mail**: b.j.norton@open.ac.uk
  - **Phone**: +44 (0) 1908 659399
# 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: +44 (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 |
<table>
<thead>
<tr>
<th>Company/Institute</th>
<th>Address</th>
<th>Contact Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institut für Informatik,</td>
<td>UIBK</td>
<td>Prof. Dieter Fensel</td>
</tr>
<tr>
<td>Leopold-Franzens Universität Innsbruck</td>
<td>Institute of Computer Science</td>
<td>University of Innsbruck</td>
</tr>
<tr>
<td></td>
<td>Technikerstr. 21a</td>
<td>A-6020 Innsbruck, Austria</td>
</tr>
<tr>
<td></td>
<td>Email: <a href="mailto:dieter.fensel@deri.org">dieter.fensel@deri.org</a></td>
<td>Tel: +43 512 5076485</td>
</tr>
<tr>
<td>ILOG SA</td>
<td>ILOG</td>
<td>Christian de Sainte Marie</td>
</tr>
<tr>
<td></td>
<td>9 Rue de Verdun, 94253, Gentilly, France</td>
<td>E-mail: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a></td>
</tr>
<tr>
<td></td>
<td>Tel: +33 1 49082981</td>
<td></td>
</tr>
<tr>
<td>inubit AG</td>
<td>inubit AG</td>
<td>Torsten Schmale, inubit AG</td>
</tr>
<tr>
<td></td>
<td>Lützowstraße 105-106, D-10785 Berlin, Germany</td>
<td>E-mail: <a href="mailto:ts@inubit.com">ts@inubit.com</a></td>
</tr>
<tr>
<td></td>
<td>Tel: +49 30726112 0</td>
<td></td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>iSOCO</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intelligent Software Components, S.A.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pedro de Valdivia 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28006 Madrid, Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +34 913 349 797</td>
</tr>
<tr>
<td>MDR Partners</td>
<td>MDR</td>
<td>Rob Davies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MDR Partners</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 St. Andrew Street, Hertford, Herts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United Kingdom, SG14 1JA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:rob.davies@mdrpartners.com">rob.davies@mdrpartners.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +44 (0)208 8763121</td>
</tr>
<tr>
<td>Hanival Internet Services GmbH</td>
<td>HANIVAL</td>
<td>Alexander Wahler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NIWA WEB Solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hanival Internet Services GmbH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kirchengasse 13/1a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A-1070 Wien</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:wahler@niwa.at">wahler@niwa.at</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +43 131 95843 11</td>
</tr>
<tr>
<td>The Open University</td>
<td>OU</td>
<td>Dr. John Domingue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Knowledge Media Institute,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Open University, Walton Hall,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Milton Keynes, MK7 6AA, UK</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>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel.: +44 1908 655014</td>
</tr>
<tr>
<td>SAP AG</td>
<td>SAP</td>
<td>Dr. Elmar Dorner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP Research, CEC Karlsruhe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP AG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vincenz-Priessnitz-Str. 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76131 Karlsruhe, Germany</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-mail: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tel: +49 721 6902 31</td>
</tr>
<tr>
<td>Company</td>
<td>Name</td>
<td>Address</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Sirma AI Ltd.</td>
<td>Sirma</td>
<td>Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Unicorn</td>
<td>Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>VUB</td>
<td>Pieter De Leenheer, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium</td>
</tr>
</tbody>
</table>
# Table of Contents

1 Introduction ................................................................. 1

2 Motivation and Aim .......................................................... 3
   2.1 Goal-driven Architectures ............................................... 3
      2.1.1 Knowledge Level Problem Solving .................................. 4
      2.1.2 Properties and Usage of Goals ..................................... 5
   2.2 Goal-driven Web Service Usage ......................................... 6

3 Requirements Analysis ....................................................... 8
   3.1 Goal-Based AI Technologies ............................................ 8
      3.1.1 Survey of Approaches and Techniques ............................... 9
      3.1.2 Summary of Findings ........................................... 12
   3.2 Requests for Web Services ............................................... 14
      3.2.1 Specifying Requested Functionalities ............................... 15
      3.2.2 Aspects Relevant for Automated Web Service Usage .............. 16
   3.3 Related DIP Technologies ............................................... 18

4 Goal Model for Web Services ............................................... 20
   4.1 Structure and General Properties ....................................... 20
      4.1.1 State-Based Model of the World ................................... 22
      4.1.2 Goal Templates and Goal Instances ................................. 23
      4.1.3 Orthogonality to Goal Resolution Techniques .................... 24
   4.2 Abstract Goals ........................................................... 24
      4.2.1 Abstract State Spaces ........................................... 24
      4.2.2 Definition .................................................. 27
      4.2.3 Relationship to SWS Frameworks ................................... 28
   4.3 Atomic Goals ............................................................. 29
      4.3.1 Definition .................................................. 29
      4.3.2 Illustrative Example ........................................... 31
   4.4 Composite Goals ........................................................ 32
      4.4.1 Definition .................................................. 33
      4.4.2 Extension of WSMO Orchestrations ................................. 34
      4.4.3 Illustrative Example ........................................... 37

5 Summary and Future Work .................................................... 39

References ........................................................................ 40

A Goal Description Ontology .................................................... 45
1 Introduction

A major concern throughout the development of technologies for automated problem solving within Artificial Intelligence (AI) is provision of more sophisticated support for the client side. End-users shall be able to formulate system usage requests on the knowledge level, i.e. specifying objectives to be achieved by stating the desire on a human understandable level and abstracting from technical details on service or resource consumption. Therefore, the concept of goals has been introduced for request formulation as a declarative objective description that abstracts from concrete service invocation. Applying this notion within IT system design yields what we refer to as a goal-driven architecture: a system takes a goal as input and determines and executes suitable resources for resolving the goal automatically.

Not surprisingly, the idea of goal-driven architectures re-emerges within the field of Semantic Web services. To enable the envisioned seamless usage of dynamically discovered and aggregated Web services, clients shall specify usage requests for Web services as goals that are resolved by automatically discovering, composing, mediating, and executing respective Web services. Out of the most prominent proposed frameworks for Semantic Web services (most notably the 4 approaches that have been submitted to the W3C as the basis for standardization efforts, chronologically OWL-S [38], WSMO [36], SWSF [9], and WSDL-S [1]), only WSMO explicitly defines Goals as a top-level notion in order to facilitate goal-driven Web service usage while the others are restricted to description models for services only.

However, the current WSMO specification defines the goal description model to consist of a requested capability and requested interfaces [52]. Conceptually, this is a mirror of the WSMO Web service description model. As this appears not to be comprehensive enough to support goal-driven Web service usage as aspired, this document presents the Goal Description Ontology GDO. This is derived from investigating existing goal-driven architectures from different AI disciplines, determining specific requirements for automated Web service usage, and integrating respective approaches from the DIP project. Apart from WSMO, the relevant DIP technologies are Web service discovery [28], the goal model developed for Web service composition [2], and goal-driven Web service invocation in execution environments like WSMX [31] and IRS [20]. Assimilating these, the central aspects of the GDO are:

1. differentiation of 3 goal types: Atomic Goals that allow specification client objectives in terms of requested functionalities and client interfaces as a means for automated Web service invocation; Composite Goals for specifying client objectives as desired workflows that can not be expressed as Atomic Goals, and Abstract Goals as the common part of Atomic and Composite Goals for specifying requested functionalities that abstracts from technical details relevant for Web service invocation

2. differentiation of Goal Templates and Goal Instances: the former denotes schematic goal definitions whereof the latter is created for expressing a concrete client objective via input instantiation; Goal Instances can only be created from Atomic or Composite Goals.
To provide a scientifically comprehensive elaboration of the goal model, the document is structured as follows: Section 2 recalls the motivation and aim of goal-driven architectures and explicates the properties of goal-driven Web service usage; Section 3 analyzes the requirements on a goal model for Web services by surveying respective approaches in related AI disciplines, identifying the relevant aspects for describing goals for automated Web service usage, and examining the related DIP technologies. On this basis, Section 4 presents the developed goal model and explicates its central properties, along with the complete meta-model layer ontology for describing goals presented in Appendix A. Finally, Section 5 concludes the document.

The presented goal model here is intended to extend, respectively replace the current WSMO goal model. In consequence, it shall be applied and adopted within execution environments for Semantic Web services developed in DIP that aim at goal-driven Web service invocation (i.e. WSMX and IRS). Furthermore, the goal model affects those mechanisms and components that deal with automated detection, usability determination, and execution of Web services whose input is provided by the client request. In particular, these are: discovery and composition, behavioral compatibility determination, and execution engines for automated Web service via a grounding to respective communication and information exchange technologies. Mediation techniques are only peripherally affected. The design of the present goal model has explicitly taken the revealed usage and requirements of dependent techniques into account.

With respect to the progress and status of the DIP project, the goal model presented here most likely will not be fully realized within the dependent DIP technologies. Rather, it provides a model derived from experiences and insights gained throughout the project that is intended to serve as a basis and input for follow-up projects that will apply and extend the DIP technology. Moreover, the elaborated goal model has been presented to the WSMO working group as a proposal for extending the WSMO model and description language in future versions of the WSMO ontology.
2 Motivation and Aim

The ultimate aim of AI research - contemporaneously its motivation and initiation in the second half of the 20th century - is to create computer systems, or, more generally, machines that can solve problems in a similar way as human beings do [61]. Several works have been and still are concerned with defining appropriate frameworks for advanced, intelligent, and automated problem solving by following the above mentioned methodologies of studying problem solving in nature and applying logic-based techniques for simulating this. A particular type of architectures for realizing the grand aim of intelligent systems emphasizes the client side. A user only specifies the objective or problem to be solved and intelligent mechanisms automatically handle the resolution process by usage of appropriate computational facilities. We refer to such systems as goal-driven architectures in the following.

While having been studied within different fields of AI research, the concept of goal-driven architectures re-emerges within Semantic Web services. In order to realize the envisioned seamless usage of dynamically discovered and aggregated Web services, clients shall specify usage requests for Web services as goals that are resolved by automatically discovering, composing, mediating, and executing respective Web services [25,50]. The aim of this work is to elaborate a generic meta-model for describing goals for automated Web service usage. As a foundation for the elaboration, the following exposes the motivation and characteristics for goal-driven architectures and explicates the intended role and usage of goals for Web services.

2.1 Goal-driven Architectures

In order to clarify the aims and scope of this document, the following exposes the origins and characteristics of goal-driven architectures. As the term ‘Goal-driven Architecture’ is rather vague and undefined in literature, we briefly explicate our understanding as well as the arising research questions on basis of an example from the Virtual Travel Agency use case scenario prominent for illustration purpose.

A client $C$ wants to book a one week holiday by using an advanced IT system $S$ that provides computational facilities for automated travel and tourism related booking. $C$ has some further constraints on the holiday package to be booked, for example: the destination should offer a beach and allow swimming in an ocean, but preferably not be in a country wherefore travel warnings are given by the government; he also wants to book a scuba diving package, and the accommodation should be located in the center of a city or village. The system $S$ should allow $C$ to specify his objective along with the constraints, while $S$ should be able to automatically detect, arrange, and utilize available computational facilities for solving the objective - similar to the service offered by a real-world travel agency.

The main merit of such systems is that they bridge the gap between the human and the machine level problem solving. While $C$ only needs to specify an objective or goal to be reached, a goal-driven system $S$ is capable of achieving this objective by automatically utilizing appropriate resources as if $C$ was dealing with another human
being. Obviously, sophisticated goal models that allow specifying user objectives and carry all information needed for automated goal resolution are the central requirement for realizing goal-driven technology. The following depicts the conceptual basis and identifies the central characteristics of goals for automated problem solving.

2.1.1 Knowledge Level Problem Solving

The concept and motivation for goal-driven architectures is allocated within Problem Solving with its theoretical-philosophical basics originating from the interdisciplinary research field of Cognitive Science.\(^1\)

Problem solving is understood as a part of human thinking concerned with how to reach an objective from the current status of the world when the procedure therefore is not known a priori. Following [7], humans apply specific psychological techniques for problem solving: discovery, notification, or observation for becoming aware of operators that can be used in a specific state, and so-called means-to-end analysis as the key mechanism for choosing the most appropriate action out of those applicable ones (roughly speaking, the current state is compared to the goal state and the difference between them is determined; then, the action is chosen that can reduce this difference to the maximum extent in comparison to others).

For simulating intelligent behavior, the aim of research in Cognitive Science is to expose the generic mechanisms that humans apply for problem solving, wherefore the fundamental theory has been provided in [47]. Although being criticized for reducing the conceptual model of mind to be presentable as an information processing system, human problem solving is defined as a goal-oriented activity for finding some possible sequence of actions that allows proceeding from the initial state of the world to the goal state. A goal is understood as a desired state that is to be reached from the current state wherefore an applicable sequence of actions is not known. An action is an activity or a process (either automated or manual) that performs the transition from one state to another in the world. Although being very basic, this model is still considered a valid theoretical basis of AI technology development. Its central aspects with regard to the motivation and design of goal-driven architectures as aspired for automated Web service usage are:

\begin{itemize}
  \item the concept of \textit{goals} as states of the world wherein a problem is considered to the solved
  \item the concept of \textit{goal-orientation}, stating that all activities are performed rationally in order to solve a goal
  \item the concept of \textit{actions} that allow changing the current status of the world (which can be performed automatically or manually)
\end{itemize}

\(^1\)Following the general AI methodology of studying structures and processes in nature as the basis for simulating them by intelligent technology, Cognitive Science has produced an impressive compilation of results that serve as the philosophic-theoretical foundation of several AI technology developments. Exhaustive synopses on the various branches and research results of Cognitive Science are provided in [63, 44], and on the Internet (e.g.: in the Stanford Encyclopedia of Philosophy \url{http://plato.stanford.edu/entries/cognitive-science/}).
• problem solving, or goal resolution, is realized by determining a possible or optimal execution sequence of actions wherefore generic strategies are applied.

Another aspect relevant for goal-driven architectures is that they are mainly concerned with the so-called knowledge level as introduced in [45]. The knowledge level is concerned with actions, effects, and behavior in the world; beneath this, the symbol level is concerned with mechanisms and operations for actually executing actions. While both levels are interconnected and both need to be addressed in order to realize goal-driven technology, the main aspects of interested within goal-driven technologies reside on the knowledge level.

Summarizing, we can understand goals and goal-driven architectures to address techniques for sophisticated client-side support for automated problem solving on the knowledge level. Goals allow specifying objectives to be achieved on a higher level of abstraction. The means for goal-driven problem solving are dynamic detection of applicable actions and operators as computational facilities for automated execution, which realize psychological methods for problem solving. Establishing the connection between the knowledge and the symbol level, goal-driven architectures can be understood as an approach for bridging the gap between human-level intelligence and automated information processing by machines.

2.1.2 Properties and Usage of Goals

With respect to the outlined characteristics of goal-driven architectures, we identify the following properties of goals and their usage for automated problem solving.

A Goal represents a formal, machine-processable knowledge level specification of a client objective that needs to satisfy the following requirements:

1. abstracting from technical details to the highest possible extent
2. support all possible kind of objectives that clients may have
3. carry all information needed for automated resolution.

A system that implements a goal-driven architecture must be capable of automatically processing such goals. The client only specifies the goal while the system automatically determines and executes the goal resolution plan by utilizing the optimal set of available computational facilities. The interaction between the client and the system for problem solving should be minimal, optimally such that the client merely submits the goal as input to the system and retrieves the result of its resolution.

In consequence, goals become the central means of human-system interaction and therewith the driving element for IT system usage. In contrast to traditional system design wherein clients need to explicitly take care of technical details for successfully utilizing the system, the concept of goal-driven architectures allow realizing the aim of “human level machine intelligence” [39]. The better a system realizes the properties of goals and their automated resolution, the better the gap between human problem solving and symbol processing by machines can be bridged.
2.2 Goal-driven Web Service Usage

The following exposes the idea for goal-driven Web service usage that realizes the properties discussed above. Figure 2.1 shows a course model the architecture. It distinguishes four top level elements for enabling automated Web service usage as goals-driven problem solving: (1) the client-side contains elements for supporting automated goal resolution from the end-user perspective, (2) the service-side contains the descriptions and implementations of operators for problem solving that typically are computational facilities, (3) intelligent mechanisms as the facilities for enabling automated goal resolution by working on the formal descriptions of the client- and the service-side, and (4) auxiliary elements needed for automated goal resolution (e.g. ontologies as formalized terminology and knowledge definitions).

Figure 2.1: Structure of Goal-Driven Architecture

The primary aspect of interest in goal-driven architectures is the client-side. This needs to provide appropriate elements for supporting problem solving from the user perspective on the knowledge level that shall be automated to the highest possible extent. The service-side needs to provide the formal descriptions of the Web services available as operators for problem solving, which commonly consist of a functional description (what the service does) and a behavioral description (how the service works, especially how to communicate with the service for consuming its functionality).

The figure distinguishes two elements for the client-side. Goals encompass the objective or problem specification by the user. It also carries the input for using Web services as automated operators for problem solving as well as constraints and preferences that the user defines for goal resolution. The Goal Resolution Plan denotes the procedure for goal resolution. It specifies when which Web service is used along with facilities for automated Web service invocation and usage. Optionally, it has a problem decomposition with respect to the available Web services. This reflects the characteristics of goal-driven architectures identified above. While the client only has to define those aspects summarized in goals - see the introductory example of booking a holiday, the goal resolution plan is determined during system runtime by automated mechanisms for detecting and aggregating available Web services.
This conception of goal-driven Web service usage is the overall approach wherefore we elaborate the meta-model for describing goals in the course of this document. While we determine the requirements for the conceptual model and the description languages the goal model in the next section, we summarize the introductory explanations with the following central aspects of relevance for technologies as aspirated here:

1. goals need to have an unambiguous formal definition for automated processing

2. the goal model must allow specifying all types of objectives or problem that clients may want to solve by using Web services

3. goals are defined on the knowledge level, i.e. abstracting from technical details for Web service usage to the highest possible extent

4. goal resolution should be automated to the highest possible extent, enforcing clients to only provide the least possible amount of information required

5. goal resolution techniques should be able to determine the optimal goal resolution plan for a given goal or problem

6. with respect to reusability of goals, the model should distinguish between general goal descriptions and concrete client objectives to be solved.
3 Requirements Analysis

This section provides the requirements analysis on specifying goals for automated Web service usage. While having depicted the aspirated architecture and usage of goals above, the central aspects of interest are (1) how goals are formally described, (2) which techniques exist for automated goal resolution and what kind of client objectives can be handled, and (3) the specific requirements arising on goal descriptions for Web services. To attain a sophisticated overview of the state of the art, we first investigate existing approaches for goal-driven architectures from different AI disciplines. Then, we examine the characteristics of Web services in order to determine the requirements on goal specifications for Web services. Finally, we discuss the realization of goal-driven Web service techniques within the DIP project.

3.1 Goal-Based AI Technologies

As outlined introductory, goal-driven architectures attend to the grand aim of Artificial Intelligence of creating intelligent systems that perform tasks automatically in a similar way that humans do. Although a commonly accepted framework for goal-driven architectures does not exist, respective approaches have been developed in several sub-disciplines of AI.

A prominent field that works with the concept of goals is Logic Programming (LP). Within PROLOG as initial and widely used LP language, the problem that is to be solved is referred to as a goal \[ \text{goal} \] \[ 13 \]. Syntactically, a goal is specified as \( ?- \phi \) with \( \phi \) as an LP expression (first-order horn logic) denoting the client objective. This can be a query such as \( ?-\text{person}(X) \) for which all facts that are persons are returned as an answer, or function calls for programs like \( ?-\text{move}(1, \text{left}, \text{right}, \text{center}) \) within the well known Tower of Hanoi puzzle. The system realizes backward-chaining and returns a term as the result, along with a success indication on the output prompt. However, a PROLOG goal is merely a query to a knowledge base or parameterized invocation of a program. In contrast, we are interested in techniques that allow specification of client objectives by abstracting from the symbol level and their resolution by automated detection and execution of available computational resources. Although PROLOG can be used to realize such functionalities, we do not consider this technology as a goal-driven architecture and concentrate on other, more specialized approaches.

Investigation reveals that the following working fields are relevant for our purpose: Cognitive Architectures that aim at realizing intelligent systems on basis of cognitive models of the human mind and intelligence \[ 6 \], Intelligent Agents that develop systems wherein autonomous agents reside that satisfy their particular objectives by interacting in a collaborative manner \[ 64 \], AI Planning that develops techniques for automatically construct plans for reaching a goal state from by subsequently applying respective operators \[ 3 \], and Knowledge Engineering that is concerned with development of techniques and systems for advanced, knowledge-based information processing \[ 60 \]. The following investigates specific frameworks and techniques for goal-driven architectures developed these disciplines.
3.1.1 Survey of Approaches and Techniques

To attain a sophisticated overview on the state of the art in goal-driven architectures, we have depicted one approach from each of the mentioned AI research fields. Referring to [58] for a detailed survey, the following briefly presents the investigated techniques and discusses the central findings. The aspects of interest for investigation are:

1. How are goals conceptually defined and formally described?
2. What techniques are applied for automated, goal-driven problem solving?
3. What kind of objectives and problems can be handled by the techniques?

SOAR

The SOAR system provides a cognitive architecture by implementing the conceptual model of human cognition presented in [46]. In essence, the SOAR technology is a production system that is claimed to represent the structure of human cognition and problem solving as determined by Cognitive Science research. The core element of the architecture are (1) so-called problem spaces that represent tasks by states, operators, and goals, and (2) a decision cycle for problem solving that chooses operators along with a learning technique for achieving a goal.

Problem spaces are state spaces that contain the domain knowledge, states, available operators, and goals of a problem. Typically, a problem space contains one goal, one or more initial states, and all operators that are applicable for the problem. As a uniform knowledge representation, all elements are described by production rules of the general form IF (condition) THEN (action): domain knowledge is described as facts (empty IF part) and rules, operators are described by a precondition in the IF part and an effect in the THEN part; goals are described as a final desired state (i.e. empty THEN part). Automated problem solving is performed by a decision cycle that iteratively repeats the following steps until the goal state is reached: (1) elaboration, i.e. all productions fire whose condition is satisfied in order to attain the current state, (2) decision, i.e. determining the operator(s) to be applied next along with preferences for them, and (3) application, i.e. the execution of the chosen operators. This is enhanced with two integrated components for intelligent problem solving. First, so-called sub-goaling as a mechanism for impasse handling: in case that there is no operator available or no clear preference can be determined, a sub-goal is defined from the current state. The resolution of this sub-goal by possibly using other problem spaces provides additional knowledge that allows to proceed within the actual problem space. The second mechanism is chunking that provides a learning facility out of solved goals and sub-goals in order to improve the system’s problem solving capabilities.

Summarizing, SOAR is a goal-driven architecture on basis of production rules and propositional logic. It supports automated problem solving for goals as final desired states in a problem space, wherein applicable operators are formally described by preconditions and effects. The problem solving technique relies on forward-chaining as the central inference technique that is enhanced with sub-goaling and learning.
BDI Agents

A central aim within the large research field of intelligent aspects is techniques for automating autonomous, rationale behavior of software agents [65]. The most prominent approach are so-called belief-desire-intention (BDI) architectures whose central aspects we summarize in the following.

The model of beliefs, desires, and intentions is a philosophical theory on the motivation and behavior of rationale action by humans presented in [14]. The three notions denote mental attitudes whose interrelations is considered to determine rationale behavior. **Beliefs** denote information on the world that an agent (regardless of being a human or a machine) considers to be true, **desires** are the objectives that the agent wants to achieve, and **intentions** are actions that the agent commits to achieve as sub-steps towards achieving a desire. An intention is defined as a **partial plan of future action that an agent is committed to execute to fulfill its desires**. This means that the agent creates a plan, i.e. a sequence of actions to be performed for achieving its objectives. An intention denotes a plan fragment that an agent considers to be beneficial for achieving its overall desire and hence commits to. After achieving an intention (i.e. the partial plan has been executed), the agent is a new state; possible changes can occur in the agent’s beliefs or desires resulting from continuative interaction with its external environment. An agent creates new intentions with respect to all beliefs in a state, commits to these and executes them. This process is repeated until a desire has been achieved and then goes on for other desires.

To allow automated reasoning on rationale behavior and collaborative interaction of agents, this model has been formalized in so-called BDI logics. The most significant logical formalizations are Cohen and Levesque’s intention logic [17] and Rao and Georgeff’s BDI logics [51]. Both logics are **modal logics with possible world semantics**, along with a **first-order logic component** for specifying epistemic and a **temporal component** for denoting dynamic aspects. All four elements are standard notions from formal logic; we refer to [65] as well as standard literature for details. The central feature of goal resolution in the BDI framework is **interleaved observation, planning, and action**. This means that in each state a BDI agent observes the world with respect to changes in the environment that possibly affect its behavioral decisions. Then, it plans the next step by determining an intention and executes this. This process is repeated until the desire is satisfied, i.e. the final objective is achieved. While this methodology is similar to the SOAR decision cycle, it is different from complete planning of the sequence of actions for goal resolution as performed by AI Planning (see below).

AI Planning

This is concerned with automated construction of **plans** as a valid sequence of actions for reaching a goal state from an initial state [30]. The classical planning problem is: given the initial state of the world, several actions and their (deterministic) effects, find a sequence of actions (viz. a plan) to achieve a certain goal state. The aim and purpose of planning is to provide techniques for dynamically combining several operators that provide smaller functionalities into an executable sequence for solving problems that require more complex functionality for resolution.
In classical planning (e.g. STRIPS [27]), objective descriptions are restricted to an initial state and a goal state to be achieved. Operators are described in terms of preconditions and effects, and plans are determined on basis of forward- or backward chaining. Plan descriptions consists of the set of operators to be applied, their ordering as the control flow, and variable bindings for the data flow. Several developments have extended the basic planning technique, most prominently efficiency increase by hierarchical task decomposition (HTN Planning, see [24]) or creation of more expressive, branching plans by conditional planning. One of the more recent approaches of interest for our work is the Extended Goal Language EaGLe that allows specifying planning goals with trial- and compensation constraints in addition to the goal state [18].

In contrast to the BDI approach of interleaved action and planning, the result of successful planning is an suitable sequence of operators for achieving a goal that is determined a priori, i.e. the complete control and data flow between used operators is determined before any of the operators is executed. Although this might cause failures in plan execution because of changes in the world that timely occur between the planning and its execution, AI Planning currently receives a renaissance as the basis for functional Web service composition (see [41, 66]).

UPML
The Unified Problem Solving Method Development Language UPML [26] is a framework for explicitly describing the reasoning behavior of knowledge based systems that work with Problem Solving Methods (PSM). PSMs are a generic problem solving methodologies that are formally described in order to allow their application for different specific problems via refinement.

UPML identifies six elements that are considered to be necessary for declaratively describing the reasoning tasks of knowledge based systems by using PSMs: (1) ontologies provide the formalized general terminology and knowledge of a domain; for specific applications, so-called (2) domain models extend ontologies with specific domain knowledge. A (3) task is a generic specification of a problem to be solved, defined by in- and output roles, the goal to be achieved as a desired state, preconditions as constraints on the initial state, and assumptions that need to hold for problem solving. A (4) problem solving method specifies a generic methodology for problem solving, described by the competence (roles, pre- & postconditions, and assumptions); complex PSMs are decomposed into sub-tasks wherefore an operational specification that defines the control- and dataflow. The other elements are (5) refiners that allow making tasks and PSMs functionally compatible by weakening or strengthening their specifications, and (6) bridges that allow resolving terminological and teleological mismatches between ontologies, domain models, tasks, and PSMs.

The overall structure of UPML co-aligns with the one for goal-driven architectures outlined in Section 2.2. Tasks denote the client side, PSMs the service-side, and the other elements refer to auxiliary elements. In contrast to the other approaches surveyed above, UPML is merely a description framework but does not encompass any goal resolution techniques. However, it provides valuable insight for advanced declarative descriptions of goal-driven systems: using ontologies as the data model allows more
expressive knowledge definitions, and the concept of tasks denotes a generic way for specifying problems or client objectives. Furthermore, when understanding PSMs as a way to generically specify goal resolution strategies rather than as operators for automated execution, their UPML specification provides an approach for describing more complex client objectives with a desired reasoning behavior.

3.1.2 Summary of Findings

As the main point of interest here, the following summarizes the central findings from the survey. We expose the types and definition of goals as well as the techniques used for automated goal resolution, and Table 3.1 summarizes the commonalities and differences of the surveyed approaches.

Goal Definitions

Regarding the first two aspects of interest, we observe that three types of problems or objectives can be supported by the examined techniques. We refer to these as Objective Types.

**Desired States of the World** (Objective Type O-I)
this denotes client objectives for creating a new object or state in the world. Examples are to buy a travel ticket by providing an origin, destination, and date as input, or the goal state of the blocks world example (i.e. the state where there is a tower of 3 blocks on the table); commonly, such goals are specified by the desired final state with respect to an initial state. Typically, we find goals of this goal type within classical planning as well as in the SOAR technology.

**Functions to be Performed** (Objective Type O-II)
this refers to objectives for performing a certain function, e.g. \(\text{multiply}(a, b)\) or \(\text{withdrawFromAccount}(x)\). While the client desire is to change the state of the world by executing a specific operation, the formal description of such goals is commonly given as a state transition, i.e. constraints on the pre- and the post-state that denote the epistemic change between them. We can find this goal type in all of the surveyed approaches.

**Temporal Abiding Goals** (Objective Type O-III)
This group denotes client objectives that remain over a longer period of time, and typically require several steps for resolution as well as adoption and goal refinement during the resolution process. Examples are to write a book or attain a PhD degree, which we typically find as desires delegated to intelligent agents.

In addition, we can differentiate two types of constraints that are supported by advanced models for goal description. We refer to them as Constraint Types.

**Goal Resolution Invariants** (Constraint Type C-I)
This denotes additional constraints that need to hold during all states of the world traversed when a goal is solved. For instance that the account shall never
become negative while a series of purchases is performed. We find such constraints implicitly hidden with the domain knowledge in the SOAR technology, as well as in BDI Agents and AI Planning.

**Goal Resolution Procedure Constraints** (Constraint Type C-II)
This refers to constraints on the process for resolving a goal. Instead of merely a black-box description for goal from an initial state to the desired goal state, requirements and constraints on intermediate steps and their order are defined in a goal formulation. We find this in form of constraints in conditional AI planning languages, as trial- and compensation specifications in the EaGLe language, and as goal decompositions in terms of collections of sub-goals with control- and data flow between them in UPML for complex PSMs.

For the formal specification of goals, we observe that all investigated approaches apply a state-based model as the underlying logical framework for formally describing goals and operators. For all three goal types, the desired goal states are defined in terms of logical expressions in some static knowledge representation language (classical logic or some ontology language). The state of the world that holds before the goal resolution procedure is started is either considered to be given as the initial state in terms of facts and rules (SOAR, BDI agents, and planning) or specified as a respective pre-state constraint (UPML). Both constraint types represent extensions of the respective goal type description. For type C-I, the constraints are commonly modelled as logical conditions in the respective framework with the meaning that these conditions need to hold during all intermediate states traversed during the goal resolution. For constraint type C-II, the constraints are either modeled as logical conditions on particular states that are required to be traversed during the goal resolution procedure (condition AI planning and EaGLe), or as declarative descriptions of goal decompositions in terms of collections of subgoals along with control- and data flow between them (operational specification of complex PSMs in UPML).

Summarizing, defining goals in terms of preconditions, effects, and additional constraints within a state-based underlying model appears to be the common characteristic of the surveyed approaches. Secondly, we can distinguish goals into *reusable templates* described by constraining formulae and *instantiations* of goals that denote concrete client objectives via variable assignment.

**Goal Resolution Techniques**
The survey has revealed three approaches for automated goal resolution techniques. The decision cycle of SOAR along with sub-goaling applies forward-chaining in order to subsequently choose the operators for reaching the goal state from the initial state. The central characteristic of automated goal resolution within the BDI framework is interleaved action and planning, meaning that the agent observes the world, then determines intentions and executes, and repeats this process until the final desire is solved. As the third one, AI Planning techniques automatically create the goal resolution plan by matchmaking and forward- or backward-chaining.
Each of these techniques allows to resolve the specific type of goals. The AI Planning as well as the SOAR technique appear to be most suitable for goal types I and II while the BDI technique in principle supports resolution for all goal types but is mainly designed and applicable for type III. As the commonalities of the goal resolution techniques, we observe that each one encompasses facilities for planning and for operator detection. The former is concerned with determining the goal resolution plan as the steps to be performed for reaching the final state from the initial state of goal formulation, and the latter is concerned with finding and choosing applicable operators out of the available ones. AI planning techniques provide the technical core as automated plan determination by finding and combing available operators. This is extended towards interleaved action and planning by sub-goaling and means-to-end analysis for operator detection in SOAR, and further into interleaved observation, planning, and action within the BDI framework.

### Table 3.1: Goal-driven Architectures - Commonalities and Differences

<table>
<thead>
<tr>
<th>Objective &amp; Constraint Types</th>
<th>SOAR</th>
<th>BDI Agents</th>
<th>AI Planning</th>
<th>UPML</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O-I, O-II C-I</td>
<td>mainly O-III C-I</td>
<td>O-I, O-II CI, C-II</td>
<td>O-I, O-II CI, C-II</td>
</tr>
<tr>
<td>Goal Description Model</td>
<td>initial state &amp; goal state in problem spaces</td>
<td>beliefs, desires, intentions</td>
<td>initial state &amp; goal state + constraints</td>
<td><strong>tasks</strong> (input &amp; output, preconditions, goal state, assumptions)</td>
</tr>
<tr>
<td>Formal Specification Language</td>
<td>production rules + propositional logic</td>
<td>BDI logics (modalities, FOL, temporal logic, possible world semantics)</td>
<td>propositional logic + plan description (operators, control- &amp; data-flow)</td>
<td>ontologies + MCL for dynamics</td>
</tr>
<tr>
<td>Goal Resolution Technique</td>
<td>forward chaining with sub-goaling</td>
<td>interleaved action / observation / planning</td>
<td>a priori plan determination</td>
<td>not in the scope</td>
</tr>
</tbody>
</table>

### 3.2 Requests for Web Services

With respect to the findings on objective types, their formal description as goals and existing techniques for automated resolution, we can now determine the requirements for defining goals for automated Web service usage. Therefore, we first inspect the general characteristics of Web services and examine existing ones in order to determine which objective types need to be supported and then identify other aspects needed for describing goals for automated Web service usage.
3.2.1 Specifying Requested Functionalities

Examining the initial idea of Web services, they have been proclaimed as a technology for enabling access to programs or computational resources via standardized interfaces over the Web [12]. The aspirated benefits are on the one hand to augment the existing Internet with access to computational resources and utilize its world-wide accessibility for enabling distributed computing over the Web [3]. With respect to the integration problem as a central motivation, the standardization of description languages of interfaces for Web service consumption is the central aspects of Web service technology [16].

Regarding the functionality provided by Web services, these can be arbitrary but are always static and passive functionalities. This means a Web service can offer any kind of functionality - from simple arithmetic calculations over more complex functionalities like combined travel booking by possibly combining several Web services up to intelligent facilities for mediation or automated coordination of system components. However, all these kinds of provided functionality are static and passive, meaning they are not dynamically changing, implemented programs instead of autonomously acting, intelligent decision systems for task delegation and automated resolution in the sense of intelligent software agents that can change the functionality during runtime [19]. Although Web services can be re-used for different problems, might be dynamically combined or achieve their functionality by on-the-fly composition and have complex interfaces that support all possible paths of interaction for service consumption, this does not change their nature of being static and passive computational resources. The concept of Semantic Web services does not change this: semantic annotations of Web services along with inference-based techniques are merely for automating or at least mechanizing Web service usage but do not change the nature and characteristics provided functionalities [40, 25, 56].

This reveals that conceptually Web services correspond to operators in the sense of the soar technology, BDI agents, and AI Planning: they provide static and passive means for programmatically performing some functionality. Thus, we conclude that the objective types relevant for Web service usage are I (desired states of the world) and II (performing a function), possibly augmented with additional constraints of type C-I or C-II. These objective types allow invoking operators in the goal-driven techniques examined above; most unlikely we will find a Web service for solving of type III objectives as long lasting desires that commonly require several sub-steps and behavioral adjustment. In consequence, a generally applicable goal model for Web services only has to support objective types I and II along with constraints of type C-I or C-II. As examined above, goals for these objective types can most suitably be described by preconditions and effects along with additional constraints within a state-based formalism as the underlying model. [ll]

[1] A comprehensive empirical study on existing Web services for verification exceeds the scope of this report. However, inspecting existing and potential Web services reveals that they commonly provide passive and static functionalities intended to be reusable for solving different problems:
3.2.2 Aspects Relevant for Automated Web Service Usage

Apart from the requested functionality as the central aspect for specifying client objectives, the aim of goal-driven Web service usage is that clients only specify the goal while its resolution is performed automatically. Therefore, a goal needs to provide all information that is required as input by the mechanisms applied for automated Web service detection and execution.

For determining the information a goal needs to encompass for automated Web service usage, we examine execution environments for Semantic Web services, with special attention to those exposing goal-driven Web service usage as illustrated in Section 2.2 (e.g. IRS [20] and or WSMX [31]). This reveals a common procedure for automated resolution of a goal as illustrated in Figure 3.1. At first, potentially usable Web services are detected out of the available ones with respect to the requested and provided functionalities. This is performed by discovery via semantic matchmaking or composition in case no directly usable Web service exists. Then, behavioral compatibility for successful interaction between the goal and the discovered or composed Web services is tested. Mediation techniques for handling possibly occurring mismatches can be utilized as auxiliary facilities. Finally, automated execution of the detected or composed Web services results in resolution of the goal.

Now, we need to examine the input information required by each component that needs to be provided within the goal description. We therefore inspect the obligatory components for discovery, composition, communication conformance testing, and execution of Web services. Data, protocol, and process mediation are auxiliary facilities that are invoked by the other components if heterogeneities need to be resolved, hence do not require additional information within goal formulations [55].

- existing Web services accessible in public UDDI repositories provide simple functionalities like currency conversion, multi-media messaging services, or information services for weather forecast or stock exchange [8].
- Web services provided by real world organizations provide generic functionalities intended to be used and combined in different applications; examples are the Amazon ECS Web service (see [35] for a semantic annotation with WSMO), or Web services provided by different shippers (see the SWS Challenge 2006 shipment scenario for a comprehensive overview: http://sws-challenge.org/wiki/index.php/Scenario:ShipmentDiscovery).
- in use case scenarios for Semantic Web services, the functionalities provided by Web services have the same characteristics; for instance booking facilities for flight, hotel, or train tickets with in VTA use case [57], business functionalities from suppliers in the DIP B2B Integration use case [29] or in the DIP e-banking use case [3], or geographical information services in the DIP e-government case study [22].
- end-user services like www.google.com or www.expedia.com - consumable via conventional Web technologies that could and might be made available as Web services in the future - mainly provide search, look-up, conversion, or booking facilities; these functionalities are computational functions performed for some input that is commonly provided via web forms.

In common, all these functionalities require goal formulations of objective type O-I or O-II as usage requests, possibly extended with constraints of type C-I or desired workflows for Web service aggregations in form of C-II constraints.
Discovery. Following the common understanding in literature, discovery is concerned with detection of directly usable Web services for solving a goal. As specified in [33] and for DIP in [28], relevant techniques for discovery are keyword-based filtering, semantic matchmaking between the requested and provided functionalities, and selection with respect to quality-of-service (QoS) information [62]. Hence, a goal description should encompass keywords and a formal specification of the requested functionality, and client-side constraints on QoS aspects.

Composition. In case no directly usable Web service is available, composition is concerned with constructing executable combinations of several Web services for solving a given goal. Within literature, we find two techniques for composition: so-called functional-level composition applies AI Planning techniques and works on the provided functionalities (e.g. [41]), while process-level composition takes the behavioral aspects described in interfaces of Web services into account (e.g. [10], see below). To attain an integrated composition technology, [49] propose to perform first functional composition that provides a skeleton of the composition which is then refined by process-level composition. As specified for DIP in [2], the techniques for candidate detection within functional composition are the same as for discovery. Moreover, client objectives may encompass conditions and constraints on the resulting composition plan. This can either constraints of type C-I or as goal decompositions as C-II constraints for specifying more complex objectives in terms of a desired process.
Communication Conformance. This is concerned with compatibility of behavioral interfaces of Web services and their clients in order to allow successful information interchange for Web service consumption [50]. As outlined in [54], this is needed for both consumption of a Web service by a client as well as for process-level composition. The former is concerned with client-service interaction, wherefore a goal description needs to encompass a behavioral interface description as the counterpart for the one the Web service to be consumed. Within DIP, this is referred to as a client interface that describes the communication structure and information interchange supported by the client for consuming a Web service [29].

Execution. As the final component for automated goal resolution by using Web services, execution is concerned with the invocation and concrete execution of the detected Web services via a grounding to communication and information exchange technologies for Web services (see DIP Invoker [43]). While concrete input data that comply with the goal description are required for invocation and execution, no additional description elements for goals arise for this component.

Concluding, we have determined that goals for Web services need to support specification of requested functionalities for goal types I and II. For enabling automated Web service usage, a goal further needs to encompass keywords and QoS constraints on the requested functionality, a client interface for Web service consumption, and provide concrete inputs for executing Web services. Not mandatory for all goals but desirable in order to allow expressing more complex objectives, a goal description model should allow specifying additional constraints for goal resolution in terms of constraints types C-I as well as desired workflows for service aggregation in form of C-II constraints.

3.3 Related DIP Technologies

Above, we have determined the requirements for a goal model for Web services. With regard to the aim of integrating and extending works from the DIP project into a coherent goal model, the following recalls the relevant DIP approaches, analyzes the commonalities and differences, and depicts the deficiencies of each.

WSMO and WSMX. WSMO is the conceptual model and foundation of the DIP technology, with WSML as the specification language and WSMX as the reference implementation and the DIP architecture [32]. In WSMO, goals are described by a requested capability and a requested interface. WSMX takes a WSMO goal as the input and solves it by dynamically detecting and executing appropriate Web services.

In principle, the current WSMO goal model allows specifying all aspects we have determined as requirements above. The requested capability can encompass keywords as QoS information as WSMO non-functional properties, and a formal functional description in terms of preconditions, assumptions, postconditions, and effects. The requested interface allows defining inputs and outputs within the state signature, a WSMO choreography description for consuming a Web service, and a WSMO orchestration description for specifying goal decompositions or constraints on the desired
workflow. However, this description model is merely a mirror of a WSMO Web service description and has been criticized in the DIP community for ambiguity of usage. Hence, we propose a refinement that integrates the following two approaches.

**IRS-III.** The Internet Reasoning Service IRS-III [20] is a broker for Semantic Web services that supports WSMO and is used as a Web service execution environment in DIP. IRS uses goals to broker between clients and Web services as follows.

A goal is a template of a requested functionality, described in terms of a WSMO capability, input and output roles, and a client interface. The latter is a special choreography interface that specifies one possible path of interaction for consuming a Web service described as a WSMO choreography with extensions for communication management [21]; in contrast, a Web service choreography interface specifies all possible paths of interaction for consuming the Web service [29]. A goal is connected to Web services that can resolve the goal via WG Mediators; this carries mediation definitions in case that any mismatch needs to be resolved, otherwise it merely binds the in- and outputs of the goal. A concrete client objective is specified by instantiating the in- and output roles of a goal; this is solved by executing the brokered Web services.

This usage of goals reveals the three aspects relevant for our context: (1) the differentiation between goals templates and instantiations for supporting goal-driven problem solving; (2) the need for input and output roles to allow instantiation and Web service invocation, and (3) the concept of a client interface for automated Web service consumption as a specialized subset of requested interfaces in WSMO goals.

**Composition Goals.** The second relevant DIP approach is the description model and language for so-called “composition goals” developed in the course of the Web service composition technique [2]. The focus of the model are goals with desired workflows (i.e. C-II constraints) to be provided as input to the composer. It is based on UML Activity Diagrams (UML-AD) and a constraint language as used also for DIP interface descriptions [59].

The model distinguishes three goal types: (1) abstract goals with in- and output roles as an abstract class for (2) atomic goals as not decomposed goals and (3) composite goals that consists of subgoals with control- and data flow definitions between them. In- and output roles are described on basis of ontologies. Several types of constraints are distinguished: so-called data-flow constraints are described by respective UML-AD constructs, while conditions in other constraint types are defined using a proprietary constraint language. The central facet of this language is easy access ontology concepts and attributes via dotted notation, i.e. `roleName.attribute operator value`; specifying such access to objects requires more complex axiom definitions in WSML. We refer to [2] for details and semantics definitions of this model and language.

The main merit of this model is that it provides an approach for specifying goal decompositions as desired workflows. This is not possible in WSMO, as an orchestration language is still under development at the time of writing. However, the approach misses formal functionality descriptions while emphasizing the need for in- and out roles in goal descriptions.
4 Goal Model for Web Services

This section presents our goal description model for enabling automated goal-driven Web service usage. As exposed in Section 2, the purpose of goals is to specify client objectives on the knowledge level along with all information needed for automated resolution by using Web services. The presented model is an integration of existing approaches along with extensions and refinements. Its design respects all requirements determined in Section 3 with special attention to the respective DIP developments. The central features are:

1. 3 distinct goal types (abstract, atomic, composite) for describing client objectives
2. Goal Instances for specifying concrete client objectives by input instantiation
3. an adequate formal definition for specifying requested functionalities
4. client interfaces for automated invocation and consumption of Web services
5. goal decomposition for specifying desired workflows.

The model is defined in generic mathematical terms independent of particular specification languages. Hence, it is adoptable to several description frameworks for Web services. For illustration, we expose the adoption to WSMO as the underlying framework of DIP. For specification, we first introduce the goal model and define its central properties, and then specify the distinct goal types in detail. The ontological definition of the goal model is provided in Appendix A of this document.

4.1 Structure and General Properties

We have determined the following requirements on goals for Web services: specify requested functionalities for objective types I (desired state of the world) and II (functions to be performed), optionally with constraints of type C-I (goal resolution invariants) and C-II (desired workflows). For automated Web service usage, a goal further needs to encompass the in- and outputs and an interface for Web service consumption. Moreover, we have depicted the differentiation of goal templates and instantiations as a desirable feature for enabling reuse and creation support of goal formulations appears. With respect to this, our goal model consists of two aspects: a meta-model for semantically describing client objectives with different goal types, and the differentiation of templates and instances of goals for the usage of goals by clients. Figure 4.1 shows the central elements and correlations of the goal model as an UML class diagram.

For the first aspect we distinguish three goal types with the following relationship. **Abstract Goals** (Section 4.2) describe client objectives in terms of preconditions and effects with respect to a domain ontology, i.e. requested functionalities independent from usage of Web service for their resolution. **Atomic Goals** (Section 4.3) extend this with explicitly defined inputs and outputs as well as client interfaces for enabling automated Web service usage, and **Composite Goals** (Section 4.4) extend Abstract
Goals with a desired workflow for goal resolution that is described as a collection of sub-goals along with control and data flow between them. All invocation and consumption of Web services is performed via Atomic Goals (also in Composite Goals). Regarding the related DIP developments, the current WSMO goal model is differentiated into three goal types. The domination and meaning of the goal types corresponds with the DIP composition goal model [2].

The second aspect of the goal model addresses reuse of goal formulations for several concrete client requests. Following the goal usage in IRS [20], we therefore define the notion of Goal Instances that denote concrete client objectives as instantiations of generic goal formulations. The three goal types consist of axiomatized description elements, i.e. the logical formulae that denote conditions with specific meaning in the meta-model. We refer these as goal templates. A goal instantiation is obtained by a variable assignment for the precondition of the goal template as the input data needed for concrete Web service invocation (see Section 4.1.2 for the detailed definition). In contrast to common logical instantiation, this requires only instantiation of a part of a goal template. Hence, our model defines Goal Instances as an explicit element. In our context, Goal Instances can only be defined for Atomic and Composite Goals as those goal types resolvable by usage of Web services.

In terms of the Meta Object Facility MOF [48] used in DIP and WSMO for defining description models, this is a meta-model (M2) layer model. Concrete occurrences of goal descriptions are located in the model-layer (M1). A specification language therefore needs to provide keywords for Atomic Goals, Composite Goals and Goal Instances with respect to their description models and correlations. While we consider the definition of a concrete goal specification language as future work in the course of adapting and integrating the goal model into WSMO and SWS frameworks, the following illustrates the conception for a Goal Instance $gi$ of an Atomic Goal $atg$. The instance refers to the template via the identifier and carries concrete input data that denote a variable assignment for the precondition of the template.

$$\text{atomicGoal} \quad \text{goaInstance}$$

$$\text{identifier} \ atg \quad \text{identifier} \ gi$$

$$\text{precondition} \ \phi \quad \text{template} \ atg$$

$$\text{effect} \ \psi \quad \text{input} \ (i_1, \ldots, i_n)$$
The following explicates the central properties of the goal model while we specify the semantic description models for the goal types in the subsequent sections.

### 4.1.1 State-Based Model of the World

In accordance to the state of the art in goal-driven architectures discussed above - as well as in accordance to rich description frameworks for Web services such as WSMO, OWL-S, and SWSF - we apply a state-based model as the underlying framework for describing goals.

The world is understood as a dynamic environment that is changed by the execution of Web services. Every state $s$ is a static snapshot of the world whereby the current state $s_c$ is defined by all facts and rules that are true at the current point of time. In general, a Web service execution results in a sequence $\tau = (s_0, \ldots, s_m)$ of state transitions in the world with $s_0$ as the start state and $s_m$ as the termination state. Similar, a goal is formulated in an initial state $s_i$ and is resolved if a state of the world $s_g$ is reached wherein the objective is achieved. Requested functionalities of goals can most suitably be described by a precondition as a state constraint for the initial state $s_i$, and an effect as a state constraint for the final desired state $s_g$. Therewith, a goal denotes a reusable objective specification; this complies with the intention of requested capabilities within the current WSMO goal model that is inspired by the concept of tasks from the UPML framework (see Section 3.1.1). We explain the formal model as well as the semantics of such functionality description in detail in the context of Abstract Goals (see Section 4.2.1).

For the following, we need to clarify the understanding and meaning of preconditions and effects. We use them as umbrella terms that encompass all aspects related to functionality descriptions. This means that the notions of preconditions and assumptions that are differentiated in WSMO capabilities are subsumed under the term preconditions in the following (the same for postconditions and effects). Although their conceptual distinction appears to be reasonable for describing Web services, there is no mathematical distinction between them. Furthermore, inputs and outputs of Web service and of goal descriptions are understood as a part of the preconditions, respectively effects. Merely required due to the technical design of Web services whose functionality can only be consumed via interfaces by providing appropriate input and retrieving respective output data, mathematically the input denotes a subset of the facts that need to be available in the initial state $s_i$ for invoking the Web service (denoted by the precondition), and the output defines a subset those facts that will be existing after successful Web service execution (denoted by the effect).

**Definition 4.1 (Precondition and Inputs, Effects and Outputs).** Let $\phi^{pre}$, $\phi^{in}$, $\phi^{eff}$, $\phi^{out}$ be formulae in a formal language $\mathcal{L}$ with model-theoretic semantics and let a functionality $\mathcal{D}_\Sigma$ be described by a 4-tuple $(\phi^{pre}, \phi^{in}, \phi^{eff}, \phi^{out})$ over some signature $\Sigma$ such that (i) $\phi^{pre}$ defines all conditions that need to hold in the initial state $s_i$, (ii) $\phi^{eff}$ defines all conditions that need to hold in the final state $s_f$, (iii) $\phi^{in}$ denotes the input for executing $\mathcal{D}_\Sigma$, and (iv) $\phi^{out}$ denotes output of executing $\mathcal{D}_\Sigma$, then $\phi^{pre} \models \phi^{in}$ and $\phi^{eff} \models \phi^{out}$. □
4.1.2 Goal Templates and Goal Instances

The second central property of the model is the distinction of goal templates and instances. Following the conception of generic task descriptions in UPML and the usage of goals in IRS (see Section 4.1.1), a goal specifies an objective that can be requested by different clients with respectively different concrete inputs and outputs.

Our model therefore defines the concept of Goal Instances that can be created for Atomic and Composite Goals. The latter are generic, re-usable specifications of a client objective such that the requested functionality as well as inputs and outputs are described as logical expressions; we refer to these as goal templates $GT$ in the following. A Goal Instance $GI_{GT}$ is an instantiation of a goal template $GT$ with concrete input data that is specified by a particular client and submitted to the system for automated resolution. The purpose of this model is that expensive operations for detecting suitable Web services are allocated on the Goal Template level such that Web service discovery and composition results for goals are re-usable. Besides, goal creation can be properly supported by user interfaces for goal search and instantiation as in IRS, thereby lifting Web service usage to the knowledge level.

Goal instantiation is achieved by defining a set of concrete literals $(i_1, \ldots, i_n)$ that satisfy the axiomatized input defined in $GT$. Formally, $(i_1, \ldots, i_n)$ denotes a variable assignment of $\phi^{pre}(GT)$ such that $(i_1, \ldots, i_n) \models \phi^{pre}(GT)$. Because of $\phi^{pre}(GT) \models \phi^{in}(GT)$ with respect to Definition 4.1, it holds that $(i_1, \ldots, i_n) \models \phi^{in}(GT)$. Hence, we refer to $(i_1, \ldots, i_n)$ the inputs of a Goal Instance $GI_{GT}$. For the formal description of Goal Instances, the input is added to the precondition of the Goal Template such that $\phi^{pre}_{GI_{GT}} = \phi^{pre}_{GT} \land (i_1, \ldots, i_n)_{GI_{GT}}$. In consequence, each Goal Instance is a semantic refinement of its Goal Template, i.e., $GI_{GT} \models GT$.

For clarification, let’s consider the following example for algebraic multiplication of two integers: the goal template $GT$ specifies $\phi^{pre} = \phi^{in} : \exists x, y. \text{integer}(x) \land \text{integer}(y)$, and $\phi^{eff} = \phi^{out} : \forall z. \text{integer}(x) \land \text{integer}(y) \land \text{integer}(z) \land z = x \times y$\footnote{The variables $x, y$ in the effect refer to the same objects as $x, y$ in the precondition, see Section 4.2.1}. Now, some client instantiates the template with the variable assignment $x = 7, y = 6$. Then, $z = 42$ is retrieved as output when invoking a suitable Web service. If some other client wants to multiply the numbers 5 and 9, he instantiates the same goal template accordingly. Hence, we can use the same Web service without need for possibly expensive discovery or composition operations. Besides, this illustrates why Goal Instances only need to instantiate the precondition: for properly invoking a Web service, concrete data for the inputs $\phi^{in}$ are needed and the preconditions $\phi^{pre}$ need to be satisfied; the concrete output as well as the effects are provided by executing the Web service used for goal solving. Definition 4.2 summarizes the above discussion.

**Definition 4.2 (Goal Instance).** Let $GT$ be a goal (atomic or composite) described by a precondition $\phi^{pre}$, an effect $\phi^{eff}$, and axiomatized input $\phi^{in}$ and output $\phi^{out}$ over some signature $\Sigma$. A Goal Instance $GI_{GT}$ is an instantiation of $GT$ such that:

(i) $\phi^{pre}_{GI_{GT}} = \phi^{pre}_{GT} \land (i_1, \ldots, i_n)_{GI_{GT}}$

(ii) $(i_1, \ldots, i_n) \models \phi^{pre}(GT)$ such that

(iii) $\forall GI_{GT}. GI_{GT} \models GT$.

---

1 The variables $x, y$ in the effect refer to the same objects as $x, y$ in the precondition, see Section 4.2.1.
4.1.3 Orthogonality to Goal Resolution Techniques

The final central aspect of the goal model is that the goal types are independent and orthogonal to the Web service techniques needed for their resolution. As those goal types whereof Goal Instances can be created, both Atomic and Composite Goals may require Web service discovery or composition along with the subsequent components for automated Web service usage identified above.

Although the reader may consider that Atomic Goals denote the goal definition for discovery and Composite Goals the one for composition, there are intuitive counter examples. Consider an Atomic Goal for multiplication of two integers: in case that there does not exist any Web services for integer multiplication but one for integer addition, then a composition of the addition Web service (i.e. iterative execution of the addition Web service) is needed. Similar, consider a Composite Goal that specifies a desired workflow for booking a flight ticket and a hotel for trip whereby the flight arrival date shall be the check-in date for the hotel (see more exhaustive discussion of this example in Section 4.4). If there exists a Web service that satisfies the workflow constraints, then discovery of this Web service is sufficient for solving the goal.

The reason is that the goal types differentiated in our model are intended to support specification of different types of objectives and not for which types of Web services or Web service usage techniques shall be used. If a potentially very complex objective can be expressed in terms of a functional description, then it can be expressed as an Atomic Goal; Composite Goals only need to be defined if constraints on the workflow are necessary for specifying the objective. The techniques needed for automated goal resolution by Web services depends on the available Web services, not on the type of client objectives specified in a goal.

4.2 Abstract Goals

The following specifies Abstract Goals in detail. Abstract Goals represent client objectives in terms of preconditions and effects abstracting from additional information for Web service usage. They serve as the common part of Atomic and Composite Goals for specifying the requested functionality; Goal Instances can not be created for Abstract Goals. We first recall the model of Abstract State Spaces that serves as the underlying formal model for describing goals. On basis of the structure and semantics of functional descriptions therein, we explicate the description structure of Abstract Goals and discuss the relationship with WSMO and other SWS frameworks.

4.2.1 Abstract State Spaces

Functional descriptions are a central element of Semantic Web service frameworks. For service advertisement, they provide a black box description of normal runs of a Web service with respect to the overall functionality provided (e.g. OWL-S Service Profiles or WSMO capabilities) or the functionality of operations for service consumption (e.g. WSDL-S or in SWSF descriptions). For goal formulation, they denote the requested functionality for specifying client objectives of type O-I or O-II.
As Web service description frameworks commonly lack of clearly defined semantics for of functional descriptions, we apply Abstract State Spaces (short: ASS) presented in [34] as a generic conceptual model with rigorous formal definitions. The ASS model assumes that Web services act in a state-based world. This is formally defined so that each state is a static snapshot of the world that is described on basis of a signature Σ and some domain knowledge Ω. The universe of the ASS world are all interpretations \( \mathcal{I}(\Sigma, \Omega) \) that are valid with regard to the signature and domain knowledge - which refers to all possible ontology instances within ontology-based applications like Semantic Web services. A statement \( \phi \) denotes a state constraint which is satisfied by a state \( s \) if there exists at least one interpretation \( \mathcal{I}(\Sigma, \Omega) \in s \) that is a model of \( \phi \). The execution of a Web service for some concrete input represents a sequence \( \tau = (s_0, \ldots, s_m) \) of states. In order to allow explicitly and properly describing functionalities, the ASS model extends the signature Σ with two types of symbols: \( \Sigma_S \) denotes static symbols that are interpreted the same way in each state \( s' \) that is a successor of \( s_0 \), and \( \Sigma_D \) denotes dynamic symbols that are interpreted in each successor state \( s' \) as in \( s_0 \). On this basis, we can define the semantics of functional descriptions for Web services and goals as follows.

**Definition 4.3.** A **Functional Description** in an Abstract State Space \( A \) is described as a 5-tuple \( D = (\Sigma_S, \Sigma_D, \phi^{\text{pre}}(\Sigma, \Omega), \phi^{\text{eff}}(\Sigma, \Omega), \phi^{\text{con}}(\Sigma, \Omega)) \) over some signature \( \Sigma \) in a language \( L \) with model-theoretic semantics such that:

1. \( \Sigma_S \) denotes static symbols such that for all \( s, s' \in S \) and \( \alpha \in \Sigma_S : \text{meaning}(s)(\alpha) = \text{meaning}(s')(\alpha) \)
2. \( \Sigma_D \) denotes dynamic symbols such that for all \( s, s' \in S \) and \( \alpha \in \Sigma_D : \text{meaning}(s_0)(\alpha) = \text{meaning}(s')(\alpha^{\text{pre}}) \)
3. \( \phi^{\text{pre}}(\Sigma, \Omega) \) is a state constraint for the initial state \( s_0 \)
4. \( \phi^{\text{eff}}(\Sigma, \Omega) \) is a state constraint for the final state \( s_m \)
5. \( \phi^{\text{con}}(\Sigma, \Omega) \) is a state constraint for all intermediate states \( s_j \in (s_0, \ldots, s_m) \)

with the meaning that if \( (s_0) \models_{L(\Sigma)} \phi^{\text{pre}} \) then \( (s_m) \models_{L(\Sigma)} \phi^{\text{eff}} \) if for all intermediate states \( s_j \in (s_0, \ldots, s_m) \) holds \( (s_j) \models_{L(\Sigma)} \phi^{\text{con}} \).

This definition gives an unambiguous, formal definition of the semantics of functional descriptions with respect to the meaning of conditions defined in a static language in a state-based world model. The model is independent of the concrete languages used for specifying constraints; the meaning of functional descriptions remains the same. We discuss the central aspects of this definition in more detail.

The first aspect is the notion of state constraints with respect to their meaning and usage. In accordance to the common definition [37], a state constraint \( \phi \) in the ASS model denotes conditions under which a state \( s \) is considered to be true, specified as a logical expression. It means that if all facts that are existing in \( s \) satisfy \( \phi \), then \( s \) is considered to satisfy \( \phi \). Formally, \( s \) satisfies \( \phi \) if all (\( \Sigma, \Omega \))-interpretations in \( s \) are models for \( \phi \), i.e.: \( \text{satisfies}(s, \phi) \leftarrow \forall x \in \mathcal{I}(\Sigma, \Omega)_A. \ x(s) \Rightarrow \text{true}(\phi) \); this can also be interpreted the other way around such that if \( \phi \) is satisfied by the existing facts in the world, then \( s \) denotes the current state of the world. Following this, preconditions and effects in functional descriptions are state constraints that need to be satisfied in the initial state and the final state of a functionality.
For Web services, preconditions constrain possible start states for execution of the Web service such that all conditions are satisfied that need to hold before the Web service can be executed, and effects constrain the states of the world that can be reached by successful execution of the Web service. For goals, preconditions constrain the initial state as states of the world when the goal is formulated or wherein its resolution shall commence, and effects constrain those states of the world in which the objective specified in the goal is considered to be achieved. For both goals and Web services, additional constraints $\phi^{com}$ are guaranteed to hold for each state of the world traversed during the Web service execution, respectively that need to hold for each state of the goal resolution process. Furthermore, Definition 4.3 only considers general state constraints, abstracting from the differentiation of computational inputs and outputs and type constraints on them. In accordance to Definition 4.1, inputs and outputs as well as all other conceptual differentiations within functional descriptions that denote syntactic subsets of preconditions and effects in the ASS model.

The second central aspect of the Definition 4.3 is the extension of symbol types in the signature $\Sigma$. In order to allow precisely describing the relationship between the pre- and post-state as well as intermediate states, the ASS model extends the signature $\Sigma$ with static symbols $\Sigma_s$ that in every subsequent are interpreted the same as in the initial state, and dynamic symbols $\Sigma_D$ whose interpretation is changed by achieving the functionality. The following gives examples for functional descriptions with static and dynamic symbols for illustration purpose with further explanations below.

Example for $\Sigma_S$: purchase contract for a product
\[
\Sigma_S : \{p\} \\
pre : \exists p. \text{product}(p). \\
eff : \exists x. \text{purchaseContract}(x) \land \text{for}(x, p).
\]

Example for $\Sigma_D$: bank account withdrawal
\[
\Sigma_S : \{a, x\} \\
\Sigma_D : \{b\} \\
pre : \exists a, b, x. \text{account}(a) \land \text{balance}(a, b) \land \text{float}(b) \land \text{float}(x). \\
\eff : \exists a, b', b, x. \text{account}(a) \land \text{balance}(a, b') \land b' = b - x.
\]

This allows to precisely describe functionalities with respect to the dependence of the post-state on the pre-state. Considering the first example, we could not formally verify that the purchase contract in the post-state is related to the product existing in the pre-state without explicitly denoting $p$ as a static symbol $\Sigma_S$. The second example describes a functionality for reducing the balance $b$ of an account $a$ by some value $x$. While we need to define $a$ and $x$ as static symbols for the same reason as above, we could not explicitly state that the amount $b$ of the account balance is changed by the functionality without declaring it as a dynamic symbol $\Sigma_D$.

Concluding, Abstract State Spaces provide a formally defined, language independent model that allows to precisely define the semantics of functional descriptions as well as other description elements of Web services. As mathematically there is no difference between the functional description of a Web service and a goal, we can beneficially apply the ASS model for describing requested functionalities in goals.
4.2.2 Definition

The following provides the definition and description model for Abstract Goals. An Abstract Goal in our model is the means for specifying requested functionalities as the common aspect of Atomic and Composite Goals. The semantics of an Abstract Goal are identical to those of functional descriptions in Abstract State Spaces as defined in Definition 4.3. Hence, Abstract Goals are language independent, meaning that it is orthogonal to and not restricted to a specific knowledge of ontology specification language. Note that Goal Instances can only be defined for Atomic and Composite Goals, while Abstract Goals are an abstract concept in our model.

**Definition 4.4 (Abstract Goal).** An Abstract Goal specifies a requested functionality as a 5-tuple $G_{\text{abstract}} = (O, \Sigma_S, \Sigma_D, \phi^{\text{pre}}, \phi^{\text{eff}}, \phi^{\text{con}})$ using a language $\mathcal{L}$ with model-theoretic semantics such that

(i) $O$ are ontologies as formal terminology and domain knowledge definitions that serve as the signature $\Sigma$ of the goal definition

(ii) $\Sigma_S$ are static symbols in the signature,

(iii) $\Sigma_D$ dynamics symbols in the signature,

(iv) $\phi^{\text{pre}}$ in $\mathcal{L}$ denotes constraints on possible initial states $s_i$ of the goal,

(v) $\phi^{\text{eff}}$ in $\mathcal{L}$ denotes constraints on final desired states $s_g$ of the goal,

(vi) $\phi^{\text{con}}$ in $\mathcal{L}$ denotes constraints that need to hold for all intermediate states $s_j \in (s_i, \ldots, s_g)$ traversed during the goal resolution.

The following specifica of Abstract Goals extend the definition of functional descriptions in the ASS model. At first, with respect to the usage of ontologies as the underlying data model of Semantic Web services - every resource description as well every data element interchanged is based on ontologies, we explicitly require usage of ontologies as the signature of an Abstract Goal. This definition considers directly used ontologies as well as terminology and domain knowledge definitions received via OO Mediators to be subsumed by $O$. Secondly, we consider preconditions and effects as mandatory description elements while additional constraints $\phi^{\text{con}}$ are optional. The reason is that all objectives of type O-I or O-II require preconditions and effects for an appropriate formal description (see Section 3.1.2), while some objectives may not require additional constraints of type C-I that are denoted by $\phi^{\text{con}}$.

As a basis for integrating the goal model into WSMO and therewith align it with DIP technologies, Listing 1 shows the description model of Abstract Goals within the WSMO framework, using the respective language for meta-model definitions [52]. We explain the differences and relationship to WSMO below in more detail.

```python
Class abstractGoal
    importsOntology type ontology
    usesMediator type \{ooMediator, ggMediator\}
    hasStaticSymbols type sharedVariable
    hasDynamicSymbols type sharedVariable
    hasPrecondition type axiom
    hasEffect type axiom
    hasConstraints type axiom

Listing 1: Abstract Goal Description Model for WSMO
```
4.2.3 Relationship to SWS Frameworks

The concept of Abstract Goals in our model correlates with the current definition of goals in WSMO. In particular, it correlates to the notion of requested capabilities that is intended to specify the requested functionality of a WSMO goal, while those aspects correlating to requested interfaces in WSMO goals is allocated within Atomic Goals in our model (see below). The similarities are that ontologies are used as the signature definition - either as directly imported ontologies or via OO Mediators, and that functionalities are specified in terms of conditions on the initial state of goal formulation and on the final desired state wherein the goal is considered to be achieved. GG Mediators are a WSMO specific element for connecting goals and denoting their relationship, thereby creating graphs of functionally related goals. Being specified elsewhere [42], these are not part of our generic goal model.

As a difference to WSMO, Abstract Goals in our model are only described by preconditions and effects while requested functionalities in WSMO are described as a capability, i.e. consisting of preconditions, assumptions, postconditions and effects.
This differentiation allows describing functionalities in a more fine grained manner: preconditions denote the conditions that need to hold before a Web service can be executed while assumptions denote additional suppositions for guaranteeing successful execution of a Web service (e.g. that the account of a credit card that is used for payment must cover the price of the purchased item). A conceptually corresponding distinction is made for post-state constraints in terms of postconditions and effects.

However, aiming at a generally applicable goal model that can be applied to different frameworks, we omit this distinction for describing requested functionalities. Moreover, formally there is no difference between preconditions and assumptions, respectively postconditions and effects in WSMO. Hence, with respect to the understanding that preconditions and effects in our model cover all aspects of functional descriptions (see Section 4.1.1), it holds that \( \phi_{G_{\text{abstract}}}^{\text{pre}} = \phi_{WSMO}^{\text{pre}} \land \phi_{WSMO}^{\text{ass}}, \) respectively \( \phi_{G_{\text{abstract}}}^{\text{eff}} = \phi_{WSMO}^{\text{post}} \land \phi_{WSMO}^{\text{eff}}. \) Hence, the distinction in WSMO capabilities appears to be of syntactic nature so that our model is compatible with WSMO with respect to the semantics of functional descriptions.

Although WSMO is the only framework for Semantic Web services that considers goals as a first class citizen in order to enable goal-driven Web service usage, we can also transfer the definition of functional descriptions from the ASS model to other frameworks. As the most prominent ones, we consider the other three approaches that as well have been submitted to the W3C as proposals for standardization. Within OWL-S [38], functional descriptions consist of inputs, outputs, preconditions, and effects (IOPE for short). These correlate as follows: \( \phi_{\text{OWL-S}}^{\text{pre}} = \phi_{\text{ASS}}^{\text{pre}} \) and \( \phi_{\text{OWL-S}}^{\text{eff}} = \phi_{\text{ASS}}^{\text{eff}}, \) and with respect to Definition 4.1.1 \( \phi_{\text{OWL-S}}^{\text{input}} \subseteq \phi_{\text{ASS}}^{\text{pre}} \) and \( \phi_{\text{OWL-S}}^{\text{output}} \subseteq \phi_{\text{ASS}}^{\text{eff}}. \) The same holds for functional descriptions of processes in SWSF [9]. For WSDL-S [1], the ASS model allows to define the intended meaning for annotating WSDL operations as follows: \( \text{precondition}_{WSDL-S} = \phi_{\text{ASS}}^{\text{pre}} \) and \( \text{effect}_{WSDL-S} = \phi_{\text{ASS}}^{\text{eff}}, \) whereby the datatype constraints of inputs and outputs are allocated within WSDL operations.

\(^2\subseteq\) denotes a semantic subset relationship in terms of Description Logics
4.3 Atomic Goals

This section specifies Atomic Goals as one of the goal types in our model wherefore Goal Instances can be defined. Atomic Goals are used for specifying all types of client objectives that do not require usage of composite goals (see next section). Apart from specifying the requested functionality, Atomic Goals carry all information that is necessary for automatically detecting and executing Web service for goal resolution. Their definition and structure correlates with the usage of goals in WSMX [31] and IRS [20]: a WSMO goal (respective an instantiation for a predefined goal) is received as input and the system detects, arranges, and invokes appropriate Web services for resolving the goal.

4.3.1 Definition

The following gives the generic definition of Atomic Goals with further explanations below. As for Abstract Goals, this definition is language independent.

**Definition 4.5 (Atomic Goal).** An Atomic Goal specifies a client objective for Web service usage as a 5-tuple $G_{atomic} = (G_{abstract}, \phi^{in}, \phi^{out}, CI, NFP)$ such that:

(i) $G_{abstract}$ specifies the requested functionality in terms of an Abstract Goal

(ii) $\phi^{in}$ is a logical formula denoting the input to be provided to a Web service with $\phi^{in} \subseteq \phi^{pre}_{G_{abstract}}$

(iii) $\phi^{out}$ is a logical formula denoting the output expected from a Web service with $\phi^{out} \subseteq \phi^{eff}_{G_{abstract}}$

(iv) $CI$ is a set of client interfaces; each describes the communication behavior of the client for invoking and consuming a particular Web service

(v) $NFP$ are non-functional properties of the goal formulation, including an item information, keywords, a natural language description, and quality-of-service requirements.

Recalling the design of the goal model, the requested functionality within an Atomic Goal is specified in terms of an Abstract Goal as defined above. In addition, an Atomic Goal specifies input and outputs for usage of Web services: inputs denote the data that the client is able or willing to provide to a Web service for invocation, while outputs constrain the computational results that are expected to be retrieved from a Web service. Both elements are defined as logical expressions, meaning that inputs and outputs are described as type constraints along with possible further conditions. The input is instantiated with a variable assignment when a Goal Instance is created for an Atomic Goal (see Section 4.1.2). In accordance to Definition 4.1 inputs are always a semantic subset of the preconditions and outputs a semantic subset of the effects of the goal description. We discuss this within the example below in more detail.

The forth description element of Atomic Goals are so-called client interfaces. Introduced for DIP in [29], this defines the communication behavior that the client can support for invoking and consuming a Web service via its respective interface. In essence, the client interface specifies the counter part of the consumption interface of a Web service. These need to be compatible so that the Web service can be consumed
correctly, whereby the creation of a client interface is the result of analyzing the requested functionality and the interface of the Web service to be used (automated or otherwise). While a Web service interface needs to specify all possible paths of interaction for consuming the provided functionality, a client interface specifies on those paths that be supported by the client.

Such a client interface is needed for each Web service that can be used to solve the goal. Hence, an atomic goal carries a set of client interfaces denoted as $CI$ in Definition 4.5. Another modelling possibility is to allocate the client interface into distinct mediators between the goal and each usable Web service that also handle other possible mismatches \cite{21}. The formal description of a client interface needs to specify the content, the direction, and the order of messages - which are the same requirements as for Web service interface descriptions. In consequence, client interfaces for goals can be described either as a WSMO choreography \cite{53}, by the DIP interface description languages \cite{59}, or by any other language that satisfies the above requirements.

An important difference to the conception of requested interfaces in the current WSMO goal description model is that a client interface in Atomic Goals only provides the counterpart for those interfaces of a Web service for consuming its functionality (i.e. WSMO choreography) but not for those that describe how the Web service achieves its functionality by using and aggregating other Web services (i.e. WSMO orchestration). The reason is that a client only needs to provide a compatible interface for consuming the Web service, whereby the implementation is not of interest. Following \cite{11}, the orchestration of a Web service denotes a refinement of the consumption interface from the client perspective. Constraints on acceptable orchestrations - e.g. usage of a certain trusted Web service for payment processing - are allocated within non-functional properties but not in the client interface.

The final description element are non-functional properties. Following the conception of WSMO, these encompass information for item management (e.g. creator, date, version, etc.), keyword descriptions (e.g. subject) and a natural language description (e.g. description) that can be used for keyword-based discovery, and quality-of-service (QoS) information (e.g. security, reliability, trust, etc.). For goal formulation, we refer to the recommendations provided in WSMO and respective extensions developed in DIP. In general for all but specifically for QoS-related aspects, a goal definition needs to encompass respective client side constraints.

For integration with the WSMO and DIP technology, Listing 2 provides the metamodel layer description for Atomic Goals within the WSMO framework. The adaption is achieved by identifying the WSMO elements that allow specifying Atomic Goals in accordance to their generic definition given above.

```
Class atomicGoal sub Class abstractGoal
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type {ooMediator, ggMediator}
  hasInput type axiom
  hasOutput type axiom
  hasClientInterface type choreography
```

Listing 2: Atomic Goal Description Model for WSMO
4.3.2 Illustrative Example

In order to illustrate the conception, definition, and usage of Atomic Goals, the following discusses an example for specifying the objective of dividing two integer numbers. Figure 4.2 provides an overview of the goal definition along with its creation and resolution procedure. As this is only intended for illustration purpose, we omit lengthy specifications for each description element but focus on the central aspects of Atomic Goals. As the specification language, we use classical first-order logic.

![Figure 4.2: Example Atomic Goal](image)

As the first step, the description of the Atomic Goal is created. It uses an ontology “numeric algebra” as that provides respective terminology definitions. The requested functionality and defines the existence of two integers as a precondition. The first one denotes the numerator, and the second one the denominator so that it cannot be equal to “0”. The effect is that there exists a positive number that represents the quotient of the integers. The inputs are two integers, and the output is a float - which in accordance to the definition are respective subsets of the precondition and effect; both are merely datatype constraints while the conditions are allocated in the precondition and the effect. Discovery of possibly usable Web services can be performed by matchmaking of the abstract requested and provided functionality descriptions.

The second step is the creation of the client interface. In order to be compatible with the “division” Web service that has been discovered, the client interface essentially denotes a counterpart of the Web service’s consumption interface. While in this example it is merely the same as the Web service interface with inverted directions of messages, in general the client interface needs to be compatible to at least one path of the Web service interface and needs to comply with possible constraints the user specifies on the goal resolution process [29].
The specification of a concrete client objective is achieved by creating a Goal Instance of the Atomic Goal. Therefore, the client specifies a variable assignment for the precondition of the Atomic Goal. In the example, the client assigns the value “7” to the numerator and “2” to the denominator. Goal Instance creation can be supported in graphical user interfaces, for instance as provided by IRS. Then, the Web service is invoked with this data as input, and returns “3.5” as the result - which complies with the output specified in the Atomic Goal.

Apart from illustrating the definition an usage of goals, this example reveals that all possibly expensive operations for Web service detection (discovery, composition, interface conformance testing) as well as goal formulation are allocated within Goal Templates. The respective results can be kept in the system and then be re-used for solving semantically equal client future requests. The only aspect of Atomic Goals not illustrated here is additional constraints $\phi^{con}$; we will discuss them below in the context of Composite Goals for clarifying the difference between these constraints and goal decompositions.

4.4 Composite Goals

As the final element of the goal model, the following specifies Composite Goals as a means for expressing complex objectives that cannot be specified as an Atomic Goal. In a nutshell, a Composite Goal decomposes a requested functionality into a collection of subgoals (that can be either Atomic or Composite Goals) along with control and data flow between them. This conception is derived from the specification of complex PSMs in UPML [26] and from the notion of composite goals that serve as the request and input for the workflow-based Web service composer specified for DIP in [2].

Composite Goals are needed for objectives whose specification requires a decomposition into subgoals in order to properly describe the process that the client requires to be performed for goal resolution. The application purpose is that a client may have an objective that consists of several steps wherefore an explicit execution order is required - for instance if a producer wants to allocate specific suppliers and shippers via Web services for particular client requests. Such objectives cannot be expressed as Atomic Goals as these merely allow defining a desired goal state to be achieved from some initial state along with constraints that need to hold in each state of the goal resolution, but no constraints on the process of goal resolution can be defined. Such objectives can be specified in terms of goal decompositions, more particular as desired processes of goals; this is what we refer to as a Composite Goal.

In this context, we recall that the goal types in our model are orthogonal to the techniques for automated Web service techniques. As discussed in Section 4.1.3 this means that Web service discovery or composition might be required for solving either goal type with respect to the available Web services. The distinction between the goal types is determined by the means of modelling needed to properly describe the client objective in formal, declarative manner.
4.4.1 Definition

As within the preceding goal type definitions, the following provides the general, language independent definition of Composite Goals. Below, we discuss the adaption to the WSMO framework.

Definition 4.6 (Composite Goal). An Composite Goal specifies a client objective for Web service usage as a goal decomposition in form of a 6-tuple $G_{\text{composite}} = (G_{\text{abstract}}, \phi^{\text{in}}, \phi^{\text{out}}, G_{\text{sub}}, \mathcal{F}, \mathcal{NFP})$ such that

(i) $G_{\text{abstract}}$ specifies the requested functionality in terms of an Abstract Goal
(ii) $\phi^{\text{in}}$ is a logical expression denoting the input to be provided to a Web service with $\phi^{\text{in}} \subseteq \phi_{\text{pre}}^{G_{\text{abstract}}}$
(iii) $\phi^{\text{out}}$ is a logical expression denoting the output expected from a Web service with $\phi^{\text{out}} \subseteq \phi_{\text{eff}}^{G_{\text{abstract}}}$
(iv) $G_{\text{sub}}$ is the set of goals that $G_{\text{composite}}$ decomposed into with $\forall g \in G_{\text{sub}}, g = G_{\text{atomic}} \lor g = G_{\text{composite}}$
(v) $\mathcal{F}$ defines the control- and data flow between the sub-goals $g \in G_{\text{sub}}$ as the order and conditions of their invocation (control aspect) and variable binding between their inputs and outputs (data aspect)
(vi) $\mathcal{NFP}$ are non-functional properties of the goal formulation, including an item information, keywords, a natural language description, and quality-of-service requirements.

Clauses (i) - (iii) define the requested functionality along with inputs and outputs, and clause (vii) defines non-functional properties. Their definition and meaning is identical to the corresponding elements of Atomic Goals in Definition 4.5, so that we omit further discussion with reference to the above explanations. Instead, we concentrate on the clauses (iv) - (vi) that define sub-goals, control- and data flow as those description elements unique for Composite Goals.

The set of sub-goals along with control- and data flow between them allows specifying client objectives in terms of desired processes of goals, with the application purpose outlined above and exemplified below in Section 4.4.3. Therefore, the first description element $G_{\text{sub}}$ denotes the set of goals that the Composite Goal is decomposed into. Referred to as sub-goals, these can be either Atomic Goals or Composite Goals. As in our model all Web service invocation is performed via the client interfaces of Atomic Goals (see Section 4.3), in general a Composite Goal can consists of several levels of goal decompositions whose lowest level consists of Atomic Goals only. For each sub-goal, the input for instantiation and the expected output is defined explicitly in the Composite Goal. The sub-goals are subsequently instantiated with concrete data when a Goal Instance for the Composite Goal is created. Therewith, the respective Web services used for solving the sub-goals can be invoked and consumed.

To properly define the desired process for achieving the sub-goals of the Composite Goal, control- and data flow are required. In accordance to the common understanding, the control flow specifies the order and conditions under which the distinct sub-goals are to be achieved. In natural language, this refers to specifications like: “first achieve $goal_1$ and then $goal_2$ if condition $c_1$ holds, otherwise do $goal_3$”. Formally, this can be
described by respective constructs in workflow-based languages, or by guarded transitions in state-based models as used within WSMO. Data flow defines the data transfer between the sub-goals, for instance in natural language: “the output of goal\textsubscript{1} serves as input for goal\textsubscript{2}”. For formally describing this, respective workflow languages provide explicit modelling constructs (e.g. in- and output pins in UML Activity Diagrams), while in state-based models variable binding is required that explicitly denotes the data flow. Although conceptually distinguished, control- and data flow are tightly interrelated. Hence, we denote them by $F$ as a compound description element in in Clause (v) of Definition 4.6.

Control- and data flow definitions between goals and Web services is supported by the DIP interface description languages \[59\] in the context of orchestration descriptions, and in particular the workflow-based description language for composition goals specified for DIP in \[2\]. Although work is ongoing at the time of writing, WSMO does not provide a completed specification for an orchestration language yet. The following therefore presents a possible extension of WSMO that has been developed in the context of DIP.

### 4.4.2 Extension of WSMO Orchestrations

The following presents an extension developed in DIP \[11\] of the WSMO choreography description language that allows specifying control- and dataflow between sub-goals and Web services aggregated in orchestrations. As this has to meet the same requirements, it can be used for specifying control- and data flow within Composite Goals in our model.

The approach is to define a minimal extension to the WSMO choreography language such that control- and data flow between goals and Web services in orchestrations can be modelled. The WSMO choreography language \[53\] is based on Abstract State Machines and defines a state signature and transition rules. These are the top-level elements for describing the consumption interface of Web services, referred to as a choreography in WSMO. The state signature defines the information used for interaction in terms of an ontology, along with on how instances of domain ontologies are used and a grounding to an existing Web service communication technology. The usage of instances is defined in terms of so-called modes that differentiated into static, in, out, shared, controlled. Transition rules specify the dynamics of choreographies in terms of guarded transitions between states. Referring to the specification for details and semantics, WSMO defines three types of rules:

- if $C$ then $R$ endIf
- $\forall V$ with $C$ do $R$ endForall
- choose $V$ with $C$ do $R$ endChoose

The central aspect of the proposed extension the so-called perform construct. This can be used in the action part transition rules for orchestration descriptions, i.e. where update rules $R$ are allocated in choreography transition rules. Intuitively, the meaning is that a resource $r$ is used as the action for state change along with the inputs provided.

---

\[3\] Here, $C$ denotes conditions defined as WSML logical expressions, $R$ defines the actions performed for changing states, and $V$ denotes variables whose scope is the transition rule such that each rule denotes a closed formula. See the WSMO choreography language specification for details on syntax and semantics \[53\].
for invocation and the outputs expected. If \( r \) is a goal, then this needs to be achieved in order to perform the state transition by executing respective Web services; if \( r \) is a Web service, then it needs to be executed. In compliance with the WSMO conceptual model, this allows to denote the Web services to be used in the orchestration either directly or indirectly; mediators are only applied in case that mismatches need to be resolved. The input and output definitions in the perform construct are logical formulae. For execution, they are instantiated with the concrete input data available in the orchestration, i.e., the inputs provided by the client of the Web service that carries the orchestration. The control- and data flow of an orchestration with perform construct can be explicitly specified by properly utilizing the different modes supported in state signature descriptions.

The following provides the definition of the perform construct along with further specifications for the adaption in WSMO. Conceptually, this correlates with the perform-construct used in the OWL-S for specifying sub-processes in the process model description of a Web service [38], which correlates with the purpose of Web service aggregations in WSMO orchestrations.

**Definition 4.7 (Perform Construct).** The *Perform Construct* specifies the invocation of goals or Web services as resources aggregated in WSMO orchestration descriptions. It is used in the action part of transition rules in addition to update rules and defined as a function \( \text{perform}(r, \phi^{in}, \phi^{out}) \) such that

(i) \( r \) denotes the resource to be invoked that can be either a goal \( G \) or a Web service \( WS \) referenced by their identifier

(ii) \( \phi^{in} \) is a logical expression denoting the input to be provided to \( r \)

(iii) \( \phi^{out} \) is a logical expression denoting the output expected from \( r \)

The perform construct allows specifying under which conditions a goal or a Web service is invoked in a WSMO orchestration, along with input- and output roles. In order to properly specify an orchestration, we also need to define the control and data flow between invoked goals and Web services. This can be achieved by properly using the modes defined in the state signature, wherefore the relevant modes are controlled, in, out, and shared: the former denotes extensions of the domain ontology concepts and relations that can only be changed inside the ASM, while the others denote ontology extensions used for communication and information interchange.

The ASMs of the orchestration and of the invoked goals or Web services are separate machines. Hence, for correct modelling, those ontology extensions that are to be provided as input to an invoked resource need to be defined with mode out or shared in the orchestration, and vice versa for outputs. Furthermore, the scope of variables in WSMO choreography descriptions is the respective transition rule. However, in order to properly define the control- and dataflow of an orchestration we need to support specifying the correspondence between distinct transition rules. Therefore, we take over the concept of shared variables as defined for WSMO capability descriptions into orchestration descriptions, such that the scope of a shared variable is the complete orchestration description and not merely one specific transition rule. With this extension, we can properly define orchestrations as collections of invoked resources along with control- and data flow between them as exemplified in Listing 35.
We abstract from concrete domain terminology in order to illustrate the above definitions. The first transition rule of this orchestration specifies that under some condition the goal $goal_1$ is invoked with an instance of concept $A$; therefore, $A$ is defined with the mode $out$ in the state signature. The result expected by $goal_1$ consist of two elements denoted by the variables $?b$ and $?c$. Then, an internal element $?d$ is created, defined by mode $controlled$. Existence of this together with the output $?b$ of $goal_1$ is the condition for invoking $ws_1$ with $?b$ as the input and some expected output $?e$. For properly specifying this, $?b$ has mode shared, and the shared variables denote the connections between the distinct transitions rules. Here, the control flow is denoted by the guards of the transition rules along with domain ontology extensions with modes $controlled$, $in$, $out$, and $shared$ in the state signature; variable binding required for data flow specification is achieved via shared variables and the state signature modes.

This extension to WSMO allows to describe the control- and data flow within Composite Goal descriptions. As the mere difference to Web service orchestration descriptions, the $perform$ construct is used to define the invocation of sub-goals only. Listing 4 provides the WSMO description meta-model for Composite Goals with WSMO orchestrations. Here, $transitionRuleExtended$ denotes the usage of the $perform$ construct in the action part of transition rules as defined above.

```
Listing 3: Example for WSMO Orchestration with invoke-constructor

Listing 4: Composite Goal Description Model for WSMO
```
We explicitly remark that the above extension to WSMO is not an official part of WSMO yet, but a proposal by the DIP community submitted to the WSMO working group for discussion.

4.4.3 Illustrative Example

For illustrating the usage and definition of Composite Goals, the following provides an example for the objective booking a flight and a hotel for a trip. The client requests a specific procedure to be followed for searching and booking. We first specify the requested functionality and then discuss the goal decomposition specifications.

We specify the requested functionality in terms of an Abstract Goal along with input and output constraints. The precondition specifies existence of all information needed as input for booking a flight and a hotel, and the effect requests existence of a flight and a hotel booking with respect to the inputs. In addition, a constraint is formulated that through the booking process the account of the credit card used for payment shall not become negative. With respect to simplicity and the focus of this example on specifying goal decompositions, we omit the specification of the used ontology and consider the in- and output definitions to be identical with the precondition and the effect.

\textbf{ontology: travel ontology.}

\[\Sigma_S = \{p, o, d, adt, rdt, c\}\]

\textbf{pre:} \[\exists p, o, d, adt, rdt, c. \text{person}(p) \land \text{city}(o) \land \text{isOrigin}(o) \land \text{city}(d) \land \text{isDestination}(d) \land \text{dateandtime}(adt) \land \text{arrivalAt}(adt) \land \text{dateandtime}(rdt) \land \text{returnAt}(rdt) \land \text{creditcard}(c).\]

\textbf{eff:} \[\forall p, o, d, adt, rdt, c \exists f, h. \text{flight}(f) \land \text{from}(f, o) \land \text{to}(f, d) \land \text{arrival}(f, adt) \land \text{return}(f, rdt) \land \text{hotel}(h) \land \text{locatedIn}(h, d) \land \text{checkin}(h, adt) \land \text{checkout}(h, rdt) \land \text{for}(f, p) \land \text{for}(h, p) \land \text{payedBy}(f, c) \land \text{payedBy}(h, c).\]

\textbf{con:} \[\forall a, b, c. \text{account}(a) \land \text{for}(a, c) \land \text{balance}(a, b) \land b > 0.\]

The client demands that the following procedure is performed for solving this goal. At first, available flights shall be found, then available hotels shall be found. Their might be special flight offers for different dates, such that the arrival date of the incoming flight determines the check-in date of the hotel, and the departure date of the return flight determines the check-out date for the hotel. Then, the client wants to personally select the particular flight and hotel to be booked. For the booking process, at first the flight shall be booked and then the hotel, with respect to that the flight availability may change in between the search and booking. Note that the general goal resolution constraint on the positive balance of the bank account is orthogonal and independent of the and requested processes of goals.

Obviously, this desired workflow can not be expressed as an Atomic Goal so that it needs to be modelled as a Composite Goal. While Figure 4.3 provides a graphical illustration of the desired workflow, Listing 5 demonstrates its specification in WSMO using the proposed extension for orchestration descriptions for the first two sub-goals “find flight” and “find hotel”.

37
For specifying a concrete client objective, a Goal Instance is created for the Composite Goal. In accordance to Definition 4.2, this is achieved by instantiating the preconditions $\phi_{G_{\text{Composite}}}^{pre}$. Subsequently, these data traverse through the subgoals so that the used Web services can be invoked and consumed automatically for resolving the Composite Goal. Hence, as for Atomic Goals the potentially expensive operations for Web service detection are allocated on the goal template level, whereby the required techniques depend on the available Web services.
5 Summary and Future Work

This document has elaborated and defined a goal model for Web services. The purpose thereof is to lift Web service usage to the knowledge level such that a client merely formulates the objective to be achieved, and submits this to the system which automatically detects and executes appropriate Web services for solving the goal.

The notion of goals and their usage as aspirated here originates from goal-driven architectures that re-emerges in the context of Semantic Web services. After exposing the aspirated architecture for goal-driven Web service usage, we have conducted a requirements analysis for defining a goal model for Web services. Investigation of existing approaches for goal-driven architectures revealed three objective types: desired states of the world (O-I), functions to be performed (O-II), and abiding goals that require several steps and adjustments for resolution (O-III). Goal formulations can be extended with constraints: invariants to hold during goal resolution (C-I), and constraints of the goal resolution procedure as desired workflows (C-II).

We have argued that Web services provide passive and static functionalities comparable to the notion of operators in earlier works. Hence, goals for Web services only need to support objectives of type O-I and O-II along with constraints of type C-I and C-II. For automated Web service usage, goals further have to carry in- and outputs as well as an interface for invoking and consuming Web services. With respect to this as well as respective developments in the DIP project - namely the WSMO goal model, the usage of goals in WSMX and IRS, and the composition goal model - we have defined a goal model for Web services.

The goal model distinguishes three goal types along with the distinction of goal templates and instances. Abstract Goals specify requested functionalities in terms of preconditions and effects along with constraints of type C-I. Atomic Goals extend this with inputs, outputs, and client interfaces as a means for automated Web service invocation, and Composite Goals realize C-II constraints by specifying desired workflows as a decomposition into sub-goals along with control- and data flow. These goal types denote generic goal formulations with axiomatized description elements. Concrete client objectives are specified by creating Goal Instances of an Atomic or Composite Goal via a variable assignment of the precondition. This allows allocating potentially complex and expensive operations for Web service detection on the goal template level as well as support for goal formulation via respective user interfaces. It is important to remark that - although our model uses similar constructs and the same languages as for Web service descriptions - goals are a means for specifying client objectives and hence are conceptually different and orthogonal to Web services.

The goal model is specified by generic mathematical definitions, hence independent of concrete specification languages so that it is applicable for different Semantic Web service frameworks. While we have outlined the adaption to the WSMO framework, the thorough integration of the goal model with definition of a specification language is future work. Because of this with respect to the progress of the DIP project, the goal model is not expected to be completely implemented in the DIP technology but rather provides a research result as a basis for future developments.
REFERENCES


A GOAL DESCRIPTION ONTOLOGY

The following provides the ontologized version of the goal model defined and elaborated in Section 4. We refer to this as the Goal Description Ontology, short GDO, that is a meta-model ontology in terms of MOF. For integration with DIP and WSMO, we use the Web Service Modeling Language WSML [15] as the specification language.

```xml
wsmlVariant _“http://www.wsmo.org/wsml/wsml−syntax/wsml−full”

namespace {“http://dip.semanticweb.org/ontologies/gdo#”,
    wsml “http://www.wsmo.org/wsml/wsml−syntax#”,
    dc _“http://purl.org/dc/elements/1.1#”
}

ontology _“http://dip.semanticweb.org/ontologies/gdo”

nonFunctionalProperties
  dc#title hasValue “Goal Description Ontology GDO”
  dc#description hasValue “A goal formally specifies a client objective along with all information required for automated Web service usage. The GDO is a meta-model layer ontology for describing goals. The goal model distinguishes three goal types — Abstract, Atomic, Composite. It introduces the concept of Goal Instances for specifying concrete client objectives by input instantiation of goals.”
  dc#subject hasValue {“MOF meta-model layer ontology”, “goal”, “Web service”, “automated usage”, “Abstract Goal”, “Atomic Goal”, “Composite Goal”}
  dc#publisher hasValue “DIP Consortium”
  dc#creator hasValue {“Michael Stollberg”}
  dc#language hasValue “en−US”
  dc#date hasValue “Date : 2006/07/04 17 : 51 : 58”
  dc#type hasValue “Revision : 1.2”

endNonFunctionalProperties

/////////////////////////////////
// Abstract Goals
/////////////////////////////////

class abstractGoal

nonFunctionalProperties
  dc#description hasValue “An Abstract Goal specifies requested functionalities as state constraints on the initial state of goal formulation (precondition) and the desired state of the world (effect) with Abstract State Spaces as the underlying formal model. Optionally, constraints can be specified that are required to hold in each state traversed for goal resolution. Abstracting from concrete input and output, this denotes the common part of Atomic and Composite Goals for specifying requested functionalities.”
  dc#relation hasValue {abstractGoalSemantics}

endNonFunctionalProperties

hasNonFunctionalProperties ofType wsml#nonFunctionalProperties

usesOntology ofType (1 *) wsml#ontology

usesOOMediator ofType (1 *) wsml#ooMediator

hasStaticSymbols ofType (0 *) wsml#sharedVariables

hasDynamicSymbols ofType (0 *) wsml#sharedVariables

hasPrecondition ofType (1 *) wsml#axiom

hasEffect ofType (1 *) wsml#axiom

hasConstraints ofType (0 *) wsml#axiom

axiom abstractGoalSemantics

nonFunctionalProperties
  dc#description hasValue “specifies the meaning of requested functionalities.”

endNonFunctionalProperties

definedBy
    ?si memberOf state and ?sj memberOf state and ?sg memberOf state and
    ?a[hasPrecondition hasValue ?pre, hasEffect hasValue ?eff, hasConstraints hasValue ?con, hasStaticSymbols hasValue ?symbS, hasDynamicSymbols hasValue ?symbD] memberOf abstractGoal and
```
meaningInState(?si, ?symbS) = meaningInState(?sj, ?symbS) and
meaningInState(?si, ?symbS) = meaningInState(?sg, ?symbS) and
meaningInState(?si, ?symbD) = meaningInState(?sg, ?symbDAtPre) and
\( \text{satisfiedInState}(\text{?si, ?pre}) \text{ and } \text{satisfiedInState}(\text{?sj, ?con}) \) implies \( \text{satisfiedInState}(\text{?sg, ?eff}) \).

concept atomicGoal subConceptOf abstractGoal
nonFunctionalProperties
dc#description hasValue "An Atomic Goal specifies client objectives in terms of requested functionalities with axiomatized input and output along with a client interface for automated Web service invocation and consumption."
dc#relation hasValue {inputOutputAsSubsets}
endNonFunctionalProperties
hasNonFunctionalProperties
usesOntology ofType wsml#nonFunctionalProperties
usesOOMediator ofType (0 *) wsml#ooMediator
hasInput ofType (1 1) wsml#axiom
hasOutput ofType (1 1) wsml#axiom
hasClientInterface ofType (1 1) wsml#choreography

axiom inputOutputAsSubsets
nonFunctionalProperties
dc#description hasValue "inputs are syntactic subset of preconditions, and outputs are a subset of the effect"
endNonFunctionalProperties

concept compositeGoal subConceptOf abstractGoal
nonFunctionalProperties
dc#description hasValue "A Composite Goal specifies client objectives as desired workflows in terms of collections of sub-goals with control- and data flow between them."
dc#relation hasValue {inputOutputAsSubsets, lowestDecompositionLevel}
endNonFunctionalProperties
hasNonFunctionalProperties
usesOntology ofType wsml#nonFunctionalProperties
usesOOMediator ofType (0 *) wsml#ooMediator
hasInput ofType (1 1) wsml#axiom
hasOutput ofType (1 1) wsml#axiom
hasSubGoal ofType (2 *) {atomicGoal, compositeGoal}
hasOrchestration ofType (1 1) orchestrationExtended

relation lowestDecompositionLevel ofType compositeGoal
nonFunctionalProperties
dc#description hasValue "all subgoals at the lowest decomposition level are Atomic Goals."
dc#relation hasValue {lowestDecompositionLevelDefinition}
endNonFunctionalProperties

axiom lowestDecompositionLevelDefinition
definedBy
!- forall {?c, ?sg} {?c memberOf compositeGoal and
?c hasSubgoal hasValue ?sg} and ?sg memberOf compositeGoal).

concept orchestrationExtended
nonFunctionalProperties
dc#description hasValue "defines WSMO orchestrations extended with the invoke construct and shared variables for specifying control— and data flow in orchestration definitions."
endNonFunctionalProperties
hasNonFunctionalProperties ofType wsml#nonFunctionalProperties
hasStateSignature ofType (1 1) wsml#StateSignature
hasSharedVariables ofType (0 +) wsml#sharedVariables
hasTransitionRules ofType (1 +) transitionRuleExtended

concept transitionRuleExtended subConceptOf wsml#transitionRule
nonFunctionalProperties
dc#description hasValue "extends the action part of transition rules for WSMO choreography descriptions with the invoke construct."
endNonFunctionalProperties
hasActionPart ofType (1 1) {wsml#updates, perform}

concept perform
nonFunctionalProperties
dc#description hasValue "allows performing goals or Web services in WSMO orchestrations."
endNonFunctionalProperties
hasResource ofType (1 1) {atomicGoal, compositeGoal, wsml#webWebservice}
hasInput ofType (1 1) wsml#axiom
hasOutput ofType (1 1) wsml#axiom

concept goalInstance
nonFunctionalProperties
dc#description hasValue "A Goal Instance specifies a concrete client objective by instantiating Atomic or Composite Goals."
dc#relation hasValue {goalInsantiation , goalInstanceTemplateRelation}
endNonFunctionalProperties
hasNonFunctionalProperties ofType (0 1) wsml#nonFunctionalProperties
hasTemplate ofType (1 1) {atomicGoal, compositeGoal}
hasInput ofType (1 1) formula

axiom goalInsantiation
nonFunctionalProperties
dc#description hasValue "a Goal Instance is created by instantiating the precondition of the goal template as a variable assignment."
endNonFunctionalProperties
definedBy
forall {?gi , ?gt} {
  (?gi memberOf goalInstance and ?gi[hasTemplate hasValue ?gt] and
   (?gt memberOf atomicGoal or ?gt memberOf compositeGoal) and
   ?gt[hasPrecondition hasValue ?preGT])
  implies ?gi[hasInput hasValue variableAssignment(?preGT)].
}

axiom goalInstanceTemplateRelation
nonFunctionalProperties
dc#description hasValue "each Goal Instance is a semantic refinement of its template."
endNonFunctionalProperties
definedBy
forall {?gi , ?gt} {?gi memberOf goalInstance and ?gi[hasTemplate hasValue ?gt] implies entails (?gi, ?gt)}. 

concept symbol
nonFunctionalProperties
dc#description hasValue "superconcept for logical symbols"
endNonFunctionalProperties

concept constant subConceptOf symbol
nonFunctionalProperties
dc#description hasValue "constants as a type of logical symbols"
endNonFunctionalProperties
concept variable subConceptOf symbol
  nonFunctionalProperties
    dc#description hasValue "variable as a type of logical symbols"
  endNonFunctionalProperties

concept term
  nonFunctionalProperties
    dc#description hasValue "denotes a term in a formal language with model—theoretic semantics"
  endNonFunctionalProperties

concept formula
  nonFunctionalProperties
    dc#description hasValue "denotes a formula in a formal language with model—theoretic semantics"
  endNonFunctionalProperties
  hasSymbol ofType symbol
    hasConnective ofType {wsml#and, wsml#or, wsml#implies, wsml#impliedBy, wsml#equivalent, wsml#neg}
    hasQuantifier ofType {wsml#forAll, wsml#exists}

relation entails (ofType formula, ofType formula)
  nonFunctionalProperties
    dc#description hasValue "denotes semantic entailment of two formulas in a formal language with model—theoretic semantics such that: 1. parameter |= 2. parameter"
  endNonFunctionalProperties

concept state subConceptOf formula
  nonFunctionalProperties
    dc#description hasValue "denotes a state of the world"
  endNonFunctionalProperties
  hasSymbol ofType {constant, term}

relation meaningInState (ofType state, ofType symbol)
  nonFunctionalProperties
    dc#description hasValue "denotes the meaning of a symbol in a state"
  endNonFunctionalProperties

relation satisfiedInState (ofType state, ofType formula)
  nonFunctionalProperties
    dc#description hasValue "defines logical satisfaction of a formula by a state of the world such that 1. parameter |= 2. parameter"
  endNonFunctionalProperties

relation variableAssignment (ofType formula)
  nonFunctionalProperties
    dc#description hasValue "denotes a variable assignment for a formula"
    dc#relation hasValue "variableAssignmentDefinition"
  endNonFunctionalProperties
  axiom variableAssignmentDefinition
    nonFunctionalProperties
    dc#description hasValue "variables in the formula are assigned with literals"
  endNonFunctionalProperties

definedBy
  forall {?f, ?v} ((?f memberOf formula and ?f hasSymbol hasValue ?v) implies ?v memberOf constant).

Listing 1: Goal Description Ontology GDO