Towards Automatic Web Service Composition using AI Planning Techniques (first draft)

Joachim Peer

August 10, 2003

Abstract

This article discusses how artificial intelligence (AI) planning techniques can be used to enable automatic composition of Web Services. Particulary, the paper discusses how standard Web Service descriptions can be annotated and converted into proper formats like PDDL to enable reasoning with modern AI planning tools.

1 Introduction

In recent years, the internet has evolved from a media of merely static data presentation to a media of interactive dynamic data and service offerings. To assist the user in making decisions about product purchases and to keep the user updated in the best possible way, intelligent adaptive systems – Personal Information Management systems (PIMs) – are required, which are able handle certain tasks on behalf of their users and deal with the complexity of the internet.

In this context, the concept of Web Services [??] looks promising, because it present services in a more machine friendly manner than traditional Web sites do; instead of graphical HTML, Web Services are built on the machine friendly XML format. According to the success of Web Services, it is very likely that PIMs will indeed primarily interact with XML based Web Services to arrange transactions over the web [11].

The user of the PIM, however, will not be interested in managing the details of the Web Service interaction, (s)he won't be interested in evaluating the purpose of a Web Service by reading its documentation or studying its WSDL description. The user will demand to be shielded off from the cumbersome aspects of Web Services, (s)he will want that these interactions are carried out in a quiet and transparent way. Specifically, the user will concentrate more on "what" needs to be done instead of "how" something needs to be done.

This clearly calls for more flexible and more independent applications than today's Web Service clients. An important part of flexibility and independence

is the agents' ability to use Web Services which are not known to the agent yet, and its ability to combine Web Services to achieve specific goals. Indeed, it is very realistic that future intelligent agents will have to fulfill a certain task by *combining* several Web Services, i.e. by constructing an individual workflow and executing it. This is the main topic of this paper and the leading research question is:

How can we describe Web Services to enable intelligent agents to automatically retrieve and compose Web Services to achieve the goals specified by their users?

Section 2 will recapitulate existing work on component and Web Service retrieval. Section 3 will provide an overview of promising work in the field of AI planning, which may be leveraged to compose Web Service executions. Section 4 will give a motivating example. It will show how information provided (or hidden!) in WSDL descriptions, can – if converted into a proper format – be used by AI planners to solve challenging problems of automatic Web Service composition. After that motivation, the discrepances between WSDL and the semiotic requirements of AI planning will be analyzed in more detail, and several proposals to enrich and translate standard Web Service descriptions will be discussed. The paper closes with a critical look on the problems left and on future work.

2 Automated Service Retrieval

When discussing Web Services, existing work on software components needs to be considered. A software component is a software element (a modular entity) which fulfills the following criteria [18]:

1. It can be used by other software elements, its "clients"
2. It comes with an official usage description, which is needed to use the component in the proper way
3. It is not tied to a specific client

It is easy to see that this definition perfectly applies to the concept of Web Services. The connection between traditional work in the area of component based Software Engineering and the new challenges of distributed computing was established by the introduction of the term "Megaprogramming" by Wiederhold et al. [29] in the early nineties. It is beneficial to recapitulate the core findings of the work on software component description and retrieval, since it provides well investigated concepts and offers insight into the theoretical underpinnings of component – and Web Service – selection.

The motivation for research in software components was spawned by the finding that the re-use of approved and tested pieces of software will lead to

a reduction in development costs and to an increase in quality. This intuitive concepts seems confirmed by the success of programming principles built on source code structuring and encapsulation like modular and object oriented programming. By using these approaches it was possible to structure software and source code; however, with the increasing amount of available pieces of software another problem evolved: the efficient storage and retrieval of software. Like all libraries, the new "software libraries" needed to be managed using some kind of description schemata and indices. Since software libraries are fully digital, another goal was to *automate* the process of component retrieval. For these purposes, at least three different approaches can be identified:

- Component retrieval as text retrieval: text retrieval algorithm are applied on textual descriptions and key words associated to software components. This kind of reasoning is not suitable for our context, since it can not guarantee sound reasoning.
- Retrieval by source code analysis: this approach builds on pioneering work by Hoare, who provided means to analyze the semantics of software by investigating its source code. However, white box based approach is in contradiction to the black box approach embraced in the area of Web Services, i.e. there is no source code available for most Web Services.
- The approach of "deductive retrieval" (also called "reuse by contract", or "design by contract") can be placed between these two extreme variants. These symbolic (non-stochastic) approaches follow the black box principle.

Of these three basic approaches, the approach of symbolic description of components is most attractive, because it opens the possibility of – at least theoretical – sound deductive reasoning. The commonly accepted basic concept behind the symbolic approach of (deductive) retrieval is the concept of the Abstract Data Type (ADT). In many cases, algebraic description style is used to denote ADTs, e.g. Larch [30], ASL [31] or the Maude syntax [4]. The signature Σ of an abstract datatype is commonly denoted by $\Sigma = (S, F)$, that is a pair consisting of a set S of sorts and a set F of operations. Further, the semantics of an ADT are specified by a set E of logical axioms, describing the properties of the datatype and its operations.

Based on these concepts, interface descriptions can be specified for software components. In virtually all approaches of component retrieval, these descriptions consist of logical statements about the inputs and outputs of a component C and logical expressions denoting its axioms E. Some approaches treat these axioms E as sets of pre-conditions Pre and post-conditions $Post$, resembling a Hoare Triple $Pre\{C\}Post$ [13]. A pre-condition specifies the state of the world which is required for a successful execution of the operation provided by C. A post-condition specifies some facts about the world which can be expected to be valid (true) after the service operation has finished its execution regularly. A component specification P can be denoted as an expression in predicate logic as follows:

$$\forall x : D_P \quad \exists z : R_P \quad I_P(x) \Rightarrow O_P(x, z) \tag{1}$$

With D_P and R_P denoting the input domain respectively output domain of a component's operation and I_P and O_P denoting the pre- respectively post-condition of the operation [21]. This can be denoted using an abbreviated syntax by integrating the statements about pre- and post-conditions with the statements about input and output types [8]:

$$(pre_Q \Rightarrow pre_C) \wedge (post_C \Rightarrow post_Q) \tag{2}$$

With Q being some specification of a certain desired functionality and C being a software component to be evaluated. If the condition is true for a specification Q and some given component C, we can assume that C will match the specification Q. We can state that the component C can be "plugged in" for all components satisfying Q, hence this condition is also called "plug in match" [32]. Beside this rather restrictive type of matching, other more relaxed condition types have been investigated, e.g. by Zaremski and Wing in [32].

Since the specifiation of the components (and their operations) in an component library $\mathcal{L}$ is generally specified in first order logic, automatic first order theorem provers can be used to test whether a certain component $C \in \mathcal{L}$ satisfies a certain component specification Q or not. This intuitive approach was proposed on many occasions, and has been re-invented several times since, but there is a serious problem with this approach: with increasing library size $|\mathcal{L}|$, an exponential growth in response time must be considered [7] [8]. In response to this problem, strategies were sought to minimize this fundamental problem of deductive component retrieval, so that the negative impact of first order reasoner's runtime behavior does not lead to unacceptable response times. In the last decade, several noteworthy approaches have been proposed in this matter:

- Load reduction by early elemination of unsuccessful candidates. Experiments [7] have shown that the response time of retrieval systems improves remarkably, if only those components, which are satisfying the given specification are forwarded to the theorem prover, and those which do not satisfy these specifications are eliminated in the process early. For instance, the NORA/HAMMR system [8] uses a configurable chain of filters to select unsuccessful candidates instead of feeding them into the FOL prover. This reduces the number of unsuccesful and time consuming proof attempts to be made by the theorem prover. Among the filters implemented in NORA/HAMMR are a filter of rejection by contradiction (using an automatic model generator) and a filter using rejecting by simplification [8].

- Load reduction by classification: Penix and Alexander [21] have proposed a heuristic approach called "feature based classification", which involves metrics to calculate the probability of a given component turning out to match a given component specification. These metrics are based on so

called "features", that are abstract relations between inputs and outputs of a component's operation. In a preparation step, the features of the components of a library can be determined and then compared to the number and type of features established for a given specification. The smaller the intersection of the feature sets of a component C and a specification Q, the little the chance of C satisfying Q.

- Reduction of the complexity of the underlying logical language: as successfully demonstrated in several areas of logic (e.g. logic programming, description logics), reduction of expressivity in logics can lead to significant improvements of their algorithmic properties. However, this requires careful balance between the expressivity necessary to provide meaningful descriptions in a certain domain and algorithmic tractability of a logical calculus.

An approach of deductive retrieval in the realm of Web Services is LARKS [25]. It allows for the description of operations by means of inputs, outputs, preconditions and effects, using terminological languages for signature specification and a rule language to denote pre-conditions and effects. Similar to LARKS is our SWS matchmaker, which was described in [20]. In contrast to LARKS, it is built on top of WSDL and it uses a slightly different (more restricted) language to express pre- and post-conditions. These approaches enable automatic service retrieval, but they do not support automatic service composition. This problem will be discussed in the following section.

3 Automated Service Composition

As in service retrieval, it makes sense to hark back to existing work carried out by other disciplines of (computer) science. Constructing a process – a plan – to attain a certain goal, i.e to get a certain task done, is a complex problem which has been investigated extensively by research in Artifical Intelligence (AI). Russel and Norvig characterize the problem of planning as follows [23]: "Planning can be interpreted as a kind of problem solving, where an agent uses its beliefs about available actions and their consequences, in order to identify a solution over an abstract set of possible plans".

A classical planning problem has the following inputs:

1. a description of the initial state of the world, denoted in some formal language
2. a description of the desired goal, denoted in some formal languages
3. a description of the possible actions which may be executed (a domain theory), again in some formal language

Some interesting AI approaches to planning are:

- Situation Calculus - a calculus of action logic [22], was developed to describe dynamic changes in the world. In Situation Calculus, it is assumed that all dynamical changes of the world are due to the execution of actions. Every situation is defined by a world history, that is a sequence of actions. The state of the world is described by functions and relations (fluents) relativized to a situation s e.g. $f(x, s)$. The constant s_0 describes the initial situation, that is a situation where no actions have occured yet. A state $do(putDown(A), do(walk(L), do(pickUp(A), s_0)))$ describes the situation created by the execution of a sequence $[pickUp(A), walk(L), putDown(A)]$.

  Golog [16] is a high-level logic programming language for the specification and execution of complex actions in dynamic domains. Golog builds on top of the situation calculus by providing a set of extra-logical constructs for assembling primitive actions (defined in Situation Calculus) into complex actions that collectively comprise a program. The application of Situation Calculus and Golog in the area of Web Services was suggested by S. McIlraith and T. Son in [17].

- Hierarchical Task Networks (HTNs) [6] - HTN planning is a method of planning by task decomposition. Contrary to other concepts of planning, the central concept of HTNs are not states, but tasks. An HTN based planning system decomposes the desired task into a set of sub-tasks, and these tasks into another set of sub-tasks (and so forth), until the resulting set of tasks consists only of atomic (primitive) tasks, which can be executed directly by invoking some atomic operations. During each round of task decomposition, it is tested whether certain given conditions are violated (e.g. exceeding a certain amount of financial resources) or not. The planning problem is successfully solved, if the desired complex task is decomposed into a set of primitive tasks without violating any of the given conditions. An approach of using HTN planning in the realm of Web Services was proposed by J. Hendler et al. [12], facilitating the planning system SHOP2.

- Graphplan, introduced by Blum and Furst in [2] and [3]: The conceptual model of Graphplan is intuitive and leads to very good performance in planning. In Graphplan, the planning problem is modeled by a graph. This planning graph consists of two types of nodes, namely action nodes and condition nodes. These nodes are arranged in layers, a layer consisting of action nodes followed by a layer of condition nodes, and so forth. Edges connect proposition nodes to the action nodes (at the next level) whose pre-conditions mention those propositions, and edges connect action nodes to subsequent propositions made true by the actions effects. Central to Graphplan's efficiency is inference regarding a binary mutual exclusion relation ("mutex") between nodes and proposition at the same level. Mutex relations help to identify potential plans by excluding impossible plans early. The process of generating potential plans is called "graph expansion". Graph expansion is repeated until a constellation is

found where a plan could be generated (i.e. necessary proposition are present and mutex relations are absent in one layer). This is a necessary (but insufficient) condition for plan existence, so Graphplan performs solution extraction, i.e. a backward chaining search to test if a plan exists in the current planning graph. While the early Graphplan algorithm as presented in [2] was restricted to propositional expressions to model the world (and the pre-conditions and post-conditions of actions), successor concepts implemented in IPP [??] or SGP [??] support modelling using predicate logic, using universal and partially even existential quantifiers.

An approach of using Graphplan for Web Service composition is presented later in this paper. We will use BLACKBOX [15], a sophisticated Graphplan based reasoner with a hybride architecture, to illustrate AI based automatic Web Service composition.

- Another important school of planning are "Constraint Satisfaction Problems" (CSP) and "Constraint Programming" [9]. The problem solving methods of CSP are integrated into many planning algorithms (e.g. the identification of mutex relations in Graphplan's can be formulated as CSP). There are several approaches which use Constrain Programming as the primary tool for plan generation, e.g. MOLGEN [24].

The desired output of a planning problem is a structure (e.g. a sequence or iteration) of actions, which can be executed in order to achieve the desired goal or state of the world, respectively. A formal language to denote planning problems is D. Mc Dermott's "Planning Domain Definition Language" (PDDL) [10], which provides an unified syntax to express planning problems of different kinds.

PDDL covers a wide range of different ways to encode planning domains and planning problems. It can serve as an umbrella for similar yet different approaches of AI planning. Since many AI planning problem solver accept PDDL as input data, it may be fruitful to try to formulate information about Web Services and agent's beliefs and desires in PDDL, in order to use existing planners to solve the problem.

According to this approach, the Web Service composition process can be depicted as follows:

As shown in Fig. 1, a planning process starts with a particular desire (goal) of an agent. The agent uses its knowledge (belief) to evaluate Web Services which might be useful to achieve the particular goal of the agent. The problem solving process consists of two main components:

1. formalizing the agent's desire, i.e. converting it to a feasible PDDL problem description

2. formalizing the available Web Services, i.e. converting the Web Service- (and, to some extent, Process-) Descriptions into PDDL domain descriptions. Of course, this includes the necessity to filter out those services which are unlikely to contribute to the agent's goal.

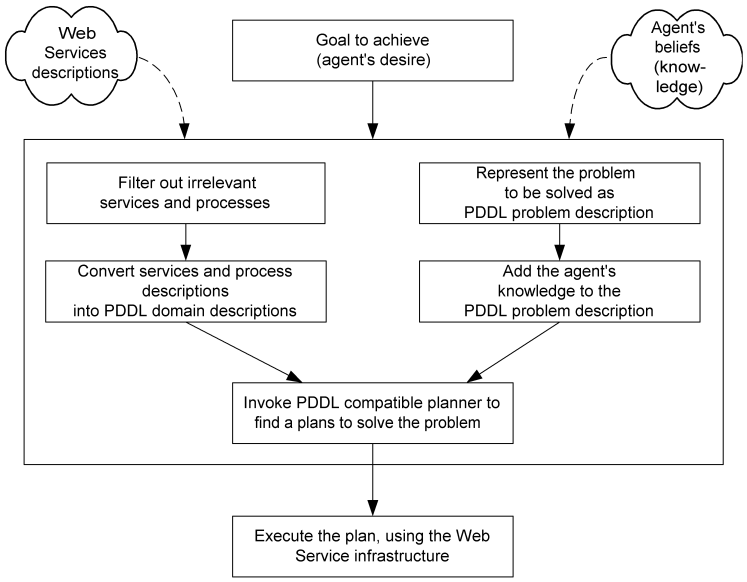

Figure 1: Process of Service Composition as planning problem

4 From Web Service Descriptions to PDDL - an Example

We start by presenting an example scenario: an agent wants to gather news from a Chinese news feed but it wants to receive the news in English, not in Chinese. While this scenario is rather simple, it can be used to illustrate some of the challenges associated with automatic service retrieval and composition.

The scenario domain consists of two Web Services. One WSDL file (cf. Listing 1) describes the Chinese news Web Service. Additionally to the Standard WSDL description, it contains a XLANG extension, which defines some "behavioral patterns" (i.e. an usage description) of the Web Service. Languages like XLANG, WSFL, BPEL4WS or even DAML-S can be used to represent such usage descriptions, and it is necessary that intelligent agents are able to interpret this information correctly to avoid errors.

```
<definitions
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  targetNamespace="http://sws.mcm.unisg.ch/ws/newsfeed.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl">
  <message name="LoginData">
    <part name="username" type="xsd:string" />
    <part name="password" type="xsd:string"/>
  </message>
  <message name="RegisterData">
```

```
  <part name="first_name" type="xsd:string"/>
  <part name="last_name" type="xsd:string"/>
  <part name="email" type="xsd:string"/>
</message>
<message name="SessionKey"
  <part name="digikey" type="xsd:string"/>
</message>
<message name="newsFeed"
  <part name="dailyNews" type="xsd:string"/>
</message>

<portType name="newsFeedPT">
  <!-- user needs to register to obtain login data -->
  <operation name="op_Register">
    <input message="RegisterData"/>
    <output message="LoginData"/>
  </operation>
  <-- user needs to log in to obtain session key -->
  <operation name="op_login">
    <input message="LoginData"/>
    <output message="SessionKey"/>
  </operation>
  <-- user needs to send a message to activate key -->
  <operation name="op_activate">
    <input message="SessionKey"/>
  </operation>
  <-- with an activated sessionID, daily news can be retrieved -->
  <operation name="op_deliverNewsFeed">
    <input message="SessionID"/>
    <output message="dailyNews"/>
  </operation>
</portType>

<binding name="NewsFeedSOAPBinding">
  <!-- details omitted -->
</binding>

<service name="NewsFeedService">
  <port name="NewsfeedPort"
   binding = "NewsFeedSOAPBinding">
     <soap:address location="http://sws.mcm.unisg.ch/ws/newsfeed"/>
  </port>

  <xlang:behavior>
    <xlang:body>
      <xlang:sequence>
        <xlang:action operation="op_login" port="NewsfeedPort" />
        <xlang:action operation="op_activate" port="NewsfeedPort" />
        <xlang:action operation="op_deliverNewsFeed" port="NewsfeedPort" />
      </xlang:sequence>
```

```
      </xlang:body>
    </xlang:behavior>

  </service>
</definitions>
```

Listing 1 - Description of a Chinese news source, including a XLANG-based usage description

The following listing presents the WSDL description of a simple Web Service for text translations from Chinese to English:

```
<definitions
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  xmlns="http://schemas.xmlsoap.org/wsdl">
    <message name="Textdata"
      <part name="textcontent" type="xsd:string"/>
    </message>

    <portType name="ChineseEnglishTranslatorPT">
      <!-- translates chinese text and returns english translation -->
      <operation name="op_Translate">
        <input message="Textdata"/>
        <output message="Textdata"/>
      </operation>
    </portType>
  </definitions>
```

Listing 2 - Description of a Web Service for linguistic translations

In the following Listing 3, we show how the knowledge about the Web Services presented above would be encoded in an AI problem description language like PDDL. The reader will notice that much of the information from the two WSDL documents listed above can be found in this description, albeit integrated and in a different shape. The reader will also notice that much of the information presented in the PDDL document can *not* be found in the original WSDL descriptions, such as information about the pre-conditions and effects of the service operations.

```
(define (domain newsfeed)
  (:requirements :strips :typing)
  (:types NEWSTEXT)
  (:constants firstname lastname email s1username s1password s1sessionid)
  (:predicates (know ?x) (language ?x ?y))

(:action S1REGISTER
  :precondition
   (and (know firstname) (know lastname) (know email))
  :effect
   (and (know s1username) (know s1password)))
```

```
(:action S1LOGIN
  :precondition
   (and (know s1username) (know s1password))
  :effect
   (know s1sessionid))

(:action S1ACTIVATE
  :precondition
   (know s1sessionid)
  :effect
   (cancall(op_delrivernewsfeed)))

(:action S1DELIVERNEWSFEED
  :parameters(?n - NEWSTEXT)
  :precondition
   (and (cancall(op_delrivernewsfeed)) (know s1sessionid))
  :effect
   (and (not (know s1sessionid))  (know ?n)  (language ?n chinese)))

(:action S2TRANSLATETEXT
  :parameters(?n - NEWSTEXT)
  :precondition
   (and  (know ?n) (language ?n chinese))
  :effect
   (and (language ?n english))))
```

Listing 3 - Web Service information encoded as PDDL domain description

Listing 4 shows the description of the software agents's problem in PDDL. The document specifies the state of the world (including the current knowledge of the agent) and specifies a goal, in terms of predicates that should become true after Web Service (or process) execution.

```
(define (problem wantenglishnewsfeed)
  (:domain newsfeed)
  (:objects
    newsitem - NEWSTEXT)
  (:init
     (know firstname)
     (know lastname)
     (know email)
     (know s1username)
     (know s1password))
  (:goal
     (and (know newsitem)  (language newsitem english)  )))
```

Listing 4 - PDDL problem description

If an agent manages to create a domain description as shown in Listing 3 and a problem description as shown in Listing 4, it can feed this information

into a PDDL compatible AI plan generator, to test whether it is possible to achieve the goal by invoking some of the available Web Services.

In case of the Graphlan based planner BLACKBOX [15], the solution found is presented as sequence of Web Service operations whose correct execution will lead to the achievement of the specified goal. In case of our example, the output of BLACKBOX looks as follows:

```
Begin plan
1 (s1login)
2 (s1activate)
3 (s1delivernewsfeed newsitem)
4 (s2translatetext newsitem)
End plan

Total elapsed time:   0.05 seconds
Time in milliseconds: 48
```

Listing 5 - Solution computed by BLACKBOX

5 How to annotate and transform WSDL

Unfortunately, PDDL based domain specifications as shown in Listing 2, can not be directly derived from standard WSDL descriptions. Instead, it is necessary to overcome the limitations of WSDL and to enrich the syntactical WSDL interface specification with semantic information, to obtain a meaningful component specification.

The semantic annotation of WSDL documents was subject to extensive work, e.g. [26], [19], [20]. We will briefly rehash the cornerstones of semantic description of WSDL documents.

5.1 Describing Pre-conditions and Post-conditions

Interface Description Languages (IDLs) like WSDL are – per definition – restricted to describe software components solely in terms of input and output signatures. It is easy to see that this kind of description is not well suited for automatic Web Service retrieval. For instance, the signature of an operation to *add* two Integer numbers is equivalent to the signature of an operation to *multiply* the numbers, that is: $Integer, Integer \rightarrow Integer$

To gather information about the intended semantics of an operation, the relationship between inputs and outputs needs to be described. In services with side effects (= real world consequences) the relationship between the state of the world before and after the service execution needs to be established.

This semantic information is usually captured in pre-condition and post-condition statements. The rationale behind the explicit formulation of pre- and post-conditions is to give software agents the opportunity to compare the semantics of desired operations with the semantics of provided operations.

The meaning of pre- and post-conditions was sketched in Sect. 2. A description of their meaning in context of Web Service operations is given below:

- A pre-condition defines requirements needed to be fulfilled by the user, in order to make a service execution possible and potentially *useful*. If the pre-condition is not fulfilled, the user (software agent) may still try to invoke the service, but it can be expected that problems will occur during service execution. For example, if a pre-condition formulates the existence of a valid user name and password, a service invocation in absence of these user credentials will result in an error message.
- A post-condition defines requirements that need to be fulfilled by the service, in order to successfully terminate the service execution. Post-conditions can be interpreted as declarative specifications of service operations, describing the consequences of the actions performed by the service. The term post-condition is synonym to the term "effect".

Traditionally, pre- and post-conditions are formulated as expressions in first order predicate logic (FOL). A FOL language is commonly constructed over a signature $\Sigma = (F, R)$, where F and R are non-empty disjoint sets of function and predicate symbols, respectively. Further, an infinite set X of variable symbols is assumed, disjointed from the symbols in Σ. As a consequence of predicate logic's known problems of undecidability and incompleteness, we need to abstain from certain features of FOL to ensure our requirement of computational tractability.

We need to chose from a number of potential language simplifications to optimize runtime behavior – a few examples are:

- To drop support for n-ary function symbols, and to forbid nested terms [20]
- To restrict expressions to Horn-like clauses [20]
- Many PDDL planners support only propositional logics, which means that no quantified variables and no predicate and function terms exist.

However, which of these simplifications serves our purposes best remains to be evaluated. In the following discussions, we will not assume any particular language restriction.

5.2 Describing Inputs and Outputs

Input and output types in WSDL documents describe the format of messages that are transferred between a Web Service and its client. As a meta language W3C XML Schema is proposed, albeit other XML Schema approaches (like Relax NG or Schematron) can also be used, as long as the clients are able to process this information.

In the realm of automatic service usage, grammar rules are not sufficient to describe inputs and outputs. Instead, agents are interested in the semantic

meaning of the inputs and outputs. That is, they need to be enabled to interpret the input and output symbols, i.e. They need to be able to make certain assumptions about the extension (interpretation) of the input and output symbols used in the Web Service description. If we assume that relevant worlds can be described as sets of objects, the extension of a symbol would refer to some specific sub-set of objects. Successful reasoning about shared information requires that those who provide and those who consume symbolic descriptions actually refer to the same (or similar) extensions of objects in a world.

This is usually achieved by using a controlled vocabulary, i.e. by introducing standards, defining terms in a way that all participants interpret them in a similar way. However, the introduction of standards is a time consuming process, which does not necessarily meet the requirements of the constantly evolving area of internet and Web Services. It is desirable that machines can interpret terms that are defined by means of logics. This topic is well studied by the research field of logic especially the emerging field of "Description Logics". Description logics [11] restrict semantic nets [68] to adhere to a controlled set of epistemological constructors ("primitives") which may be used as building blocks defining complex structures (ontologies [30]). Among the most expressive, yet tractable, members of the Description Logic families is the $\mathcal{SHIQ}$ logic, which is implemented by Description Logic engines like FaCT [14] and RACER [28]. A modified version of $\mathcal{SHIQ}$ served as starting point for the development of Semantic Web based ontology languages DAML+OIL [1] [27] and its successor OWL [5]. These logics are the latest outcome of the efforts to create a global web of machine readable knowledge.

Inputs and outputs can be integrated into PDDL pre-conditions and post-conditions. An input can be interpreted as part of a pre-condition. To state that literal I is an input of an operation, we can declare that the predicate $know(I)$ is true[1]. Outputs can be expressed in terms of post-conditions. To state that O is an output of an operation, we simply attach a post-condition $know(O)$ to the operation.

5.3 Translating Web Service Usage descriptions

This section contains some errors, which will be removed in the second draft of this paper. Any comments for more elegant solutions are welcome

As discussed above, process descriptions are often used to describe how users need to interact with Web Services in order to achieve useful results. The most important constructs for such usage descriptions are sequence, choice and iteration. The following paragraphs will illustrate how these constructs can be translated into PDDL.

[1] in case of propositional logic, we would chose a literal instead of a predicate

5.3.1 Sequence

Sequences of operations can be expressed by pre- and post-conditions. We need to introduce a set of sentences which represent whether an agent may call a certain operation or not.

A sentence $maycall(op_x)$ indicates that the agent is allowed to invoke operation op_x, whereas $\neg maycall(op_x)$ indicates that the agent may not invoke this operation. The logical sentence $maycall(op_x)$ can be expressed using predicates and constants or simply by propositional boolean sentences.

The sequence $\langle op_1, op_2, op_3 \rangle$ can be expressed by means of pre- and post-conditions as follows:

```
(:action op1
   :effect (maycall op2))
(:action op2
   :precondition (maycall op2)
   :effect (maycall op3))
(:action op3
   :precondition (maycall op3))
```

The predicate maycall is used to signal whether an agent should (may) call a certain operation or not.

5.3.2 Choice

Choice constructs like the BPEL4WS *switch* operator are used to distinguish between two or more different paths a service execution may take. There are two different types of choices: deterministic (transparent) and indeterministic (opaque) ones. We will only consider deterministic choices here.

A deterministic choice $\langle if\ A\ then\ \langle op_1 \rangle else\ \langle if\ B\ then\ \langle op_2 \rangle else\ \langle if\ C\ then\ \langle op_3 \rangle \rangle \rangle \rangle$ can be translated into PDDL as follows:

```
(:action op1
   :precondition (and A (not B) (not C)))
(:action op2
   :precondition (and B (not A) (not C)))
(:action op3
   :precondition (and C (not A) (not B)))
```

TO DO: to correctly reflect the semantic of the switch construct we need to create aliases and append post-conditions to make sure that only one and not several op's are invoked

5.3.3 Iteration

Iteration constructs like the *switch* operator of XLANG and BPEL4WS, can also be translated to fit the requirements of AI planning tools. A statement $while(A)\ \langle blockB \rangle$ can be translated to PDDL as follows. Each operation $op_n \in$

B is assigned an alias name, e.g. $op_{b-1}, op_{b-2}, ..., op_{b-n}$. For each of these alias names, the condition A is attached as pre-condition. This tells the planner that the aliased operations (those operations that belong to the while-block) can only be performed if condition A is true. This does not impose restrictions to other contexts, i.e. outside of the while loop.

```
(:action op1-b
   :precondition (and A (...)))
(:action op2-b
   :precondition (and A (not A) (...)))
```

6 Open Problems and Future Work

Among the issues that need to be clarified by future work are the following questions:

- Which restrictions do we have to impose on FOL statements to describe pre- and post-conditions in an reasonable expressive yet algorithmic tractable manner?
- Which set of "reserved words" (predicates, constants) do we have to define for the PDDL domain defintions? In this paper we presented the predicates *know* and *maycall*, but many other reserved words will be turn out to be useful for semantic Web Service description.

We are planning a project to test and compare several of these possibilities empirically. We hope this will contribute to the clarification of these issues.

7 Summary and Conclusion

In this paper we proposed a concept for the semantic description of Web Services. We have shown how WSDL documents can be annotated by semantic information, and we have illustrated how this information can be transformed into PDDL, a format used by AI planners.

References

[1] DAML+OIL language specification.

[2] Blum, A., and Furst, M. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)* (1995), pp. 1636–1642.

[3] Blum, A., and Furst, M. L. Fast planning through planning graph analysis. *Artificial Intelligence*, 1-2 (1997).

[4] Clavel, M. e. a. A maude tutorial, 2002.

[5] Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. OWL web ontology language 1.0 reference.

[6] Erol, K., Hendler, J., and Nau, D. S. Semantics for hierarchical task network planning, 1994.

[7] Fischer, B. Deduction based software component retrieval, 2001.

[8] Fischer, B., and Schumann, J. NORA/HAMMR: Making deduction-based software component retrieval practical. In *Proc. CADE-14 Workshop on Automated Theorem Proving in Software Engineering* (1997).

[9] Fruehwirth, T., and Abdennadher, S. *Constraint Programmierung*. Springer Lehrbuch. Springer Verlag, Berlin, Heidelberg, 1997.

[10] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. PDDL—the planning domain definition language. In *AIPS-98 Planning Committee* (1998).

[11] Hendler, J. Is there an intelligent agent in your future. *Nature, Web Matters* (1999).

[12] Hendler, J., Wu, D., Sirin, E., Nau, D., and Parsia, B. Automatic web services composition using Shop2. In *Proceedings of The Second International Semantic Web Conference(ISWC)* (2003).

[13] Hoare, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM*, 10 (1969).

[14] Horrocks, I. Using an expressive description logic: FaCT or fiction? In *Prof. of KR'98* (1998).

[15] Kautz, H., and Selman, B. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Workshop on Planning as Combinatorial Search, AIPS-98, Pittsburgh, PA, 1998* (1998).

[16] Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 1-3 (1997).

[17] McIlraith, S., and Son, T. Adapting Golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)Toulouse, France, April 2002* (2002).

[18] Meyer, B. The grand challenge of trusted components. In *Proc. of the 25th International Conference on Software Engineering, May 03 - 10, 2003 Portland, Oregon* (2003), IEEE.

[19] Peer, J. Bringing together semantic web and web services. In *Proceedings of The First International Semantic Web Conference (ISWC)* (2342 2002), I. Horrocks and J. Hendler, Eds., no. 2342 in Lecture Notes in Computer Science, Springer-Verlag.

[20] Peer, J. Semantic annotation and matchmaking of web services. In *Submitted to: Workshop on Semantic Web and Databases at Very Large Databases Conference, VLDB-2003* (2003).

[21] Penix, J., and Alexander, P. Efficient specification-based component retrieval. *Automated Software Engineering* (1999).

[22] Pratt, V. Action logic and pure induction. In *Proc. Logics in AI: European Workshop JELIA '90* (1990), J. van Eijck, Ed., Springer-Verlag Lecture Notes in Computer Science, pp. 97–120.

[23] Russel, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice-Hall Inc., 1995.

[24] Stefik, M. Planning with constraints. *Artificial. Intelligence* (1981).

[25] Sycara, K., Widoff, S., Klusch, M., and Lu, J. LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 2 (2002).

[26] The DAML-S Coalition. DAML Web Services V0.9, 2002.

[27] van Harmelen, F., Patel-Schneider, P. F., and Horrocks, I. A model-theoretic semantics for DAML+OIL, 2001.

[28] Volker Haarslev, R. M. Description of the racer system and its applications. In *Proceedubgs International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August 2001* (2001).

[29] Wiederhold, G., Wegner, P., and Ceri, S. Toward megaprogramming. *Communications of the ACM*, 11 (1992).

[30] Wing, J. M., Rollins, E., and Zaremski, A. M. Thoughts on a larch/ML and a new application for LP. In *Proceedings of the First International Workshop on Larch* (1993), U. Martin and J. Wing, Eds., Springer-Verlag, pp. 297–312.

[31] Wirsing, M. Structured algebraic specifications: A kernel language. *Theoretical Computer Science* (1986).

[32] Zaremski, A. M., and Wing, J. M. Specification matching of software components. In *Proceedings of 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering* (1995).