Web Service Discovery and Composition for Virtual Enterprises

Jürgen Dorn, Peter Hrastnik, and Albert Rainer
ec3 - electronic commerce competence center
{juergen.dorn, peter.hrastnik, albert.rainer}@ec3.at

ABSTRACT:

One main characteristic of virtual enterprises are short-term collaborations between business partners to provide efficient and individualized services to customers. The MOVE project targets at a methodology and a software framework to support such flexible collaborations based on process oriented design and communication by Web services. MOVE's framework supports the graphical design and verification of business processes, the execution and supervision of processes in transaction-oriented environment, and the dynamic composition and optimization of processes. A business process may be composed from a set of Web services, deployed itself as Web service and executed in the framework. The composition of processes from Web services is implemented with methods from AI-planning. We apply Answer Set Programming (ASP) and map Web service descriptions and customer requests into the input language of the ASP software DLV. Composition goals and constraints guide a composition challenge. We show the performance of our program and give some implementation details. Finally we conclude with some insights.

KEY WORDS:
Keywords: Web Service Composition, AI planning, Answer Set Programming, Virtual Enterprise

INTRODUCTION

A *virtual enterprise* is a business model (with many variants) in which different legally independent companies cooperate electronically (mainly by means of the Internet) to offer better services to customers that cannot be offered by one single member. The configuration of new services for customers is faster and more flexible than in traditional business models because a cooperation is initiated electronically.

Since one of our industrial partners comes from the tourism sector and another partner is an Austrian mobile communication provider, we use scenarios from the tourism sector extended with mobile communication features. In the tourism sector, partners in a virtual enterprise are for instance airlines, taxi drivers, hotels and others. In particular, the better coordination of services (e.g., flight arrival, luggage transport, taxi drive and hotel check-in) is an advantage for customers achieved by sophisticated information and communication technology in a virtual enterprise. Moreover, the possibility to contact the customer anytime and anywhere by means of mobile communication can result in continually better services. Of course, there are also benefits for single members of the virtual enterprise (e.g. the taxi driver), because they can schedule their workload better.

Mainly small enterprises exist in the tourism sector. These can participate in the development of larger products – so called *dynamic holiday packages* with the virtual enterprise concept. In contrast to actual business practices between tour operators and hotels, the virtual enterprise concept leads to more flexibility in defining such packages. A small enterprise can define a new service, announce it in the virtual enterprise and a customer can select this service immediately.

It is assumed that the cooperation is often short-term because the collaboration is established only for one individual service. Thus, the establishment of a collaboration must be achievable very easily without great organizational or financial efforts, but it must conform to certain standard behavior in order to guarantee high quality. For example, let us consider that the luggage of an airline customer is automatically transported to the hotel by taxi without requiring the customer to take care of it. Since the virtual enterprise has all required data, the taxi is already waiting for the customer and the luggage is immediately transported from the aircraft to the taxi. This process has to be controlled and supervised electronically to avoid the luggage loss. An architecture based on standardized Web services is a promising technique for achieving the required flexibility as well as the quality. The interfaces between members of the virtual enterprise (i.e. the data transmitted) are implemented on basis of international standards, especially those based on XML. Furthermore, open source software is reused as much as possible in the Move-project. Hrastnik (2004) and Rainer (2004) describe these infrastructure aspects in more detail.

Members of a virtual enterprise do not commit themselves to any financial or legal obligations. However, each member has to express which services it will supply under which conditions to the virtual enterprise and for the end customer in a transparent way. Some central decision support system to enable fast and flexible reactions to customer queries has to exist. The ideal solution would be an automatic decision based on centrally stored knowledge as well as knowledge distributed over all partners accessible through electronic communication. However, there may also be problems that require human expert knowledge. There will be a variety of commitments from members to such a decision support system.

Given a Web services based architecture, we can define a planning component as one Web service that obtains the customer query as input and produces a document containing several proposals as output. If the user selects one proposal, a second Web service can reserve and book this proposal. The first Web service calls further Web services to obtain the customer's profile data and preferences stored somewhere in the Internet. The second Web service may store an adapted version of the customer's preferences.

The planner may search for resources in the Internet by querying further Web services published in repositories or other sources. Thus, the planner may find available rooms in hotels and seats in aircrafts. If the customer searches for complex services (dynamic holiday packages), the following prototypical process steps are executed:

1) configuration step: define which basic services / parts (including certain attributes) should be part of the final service / product
2) coordination step: coordinate and refine the parts (temporal, geographical or qualitative fitting)
3) resource instantiation step: check the availability of resources and assign resources to services
4) reservation step: resources are reserved to enable a consistent complex product
5) customer feedback step: ask customer for acceptance (or rejection)
6) booking step: book the accepted services, if desired
7) adaptation step: sometimes a booking change is desired

In order to define the basic services, the system requires some basic model, which describes the concept *holidays* or *travel*. Such a model describes required and optional parts. The customer may select from several alternative models. We assume the customer has made the first specification of his request, such as "family winter holiday over Christmas". In general, the exact period for the holidays is not fixed by hard constraints. Sometimes hotels are available only for a shorter duration or a flight ticket might be cheaper on a certain day. In these cases the system must be able to propose solutions fitting the customer's general specification in the best possible way. Perhaps, the customer has filled out additionally a questionnaire about his profile (address,

gender, etc.). This is not part of the planning process. Further, he may have specified explicit preferences (e.g. likes sports, likes theatre, etc.). Finally, a record of earlier bookings may suggest some implicit preferences (e.g. cheap holidays or high quality, etc.). A destination can be proposed from customer's preferences and hence, the transportation may be derived from the customer's home address and the proposed destination. Further, the system may propose special events in accordance with the customer's preferences (e.g. a golf tournament). Often, customers do not want to book such events so early. However, it may be that the customer will stay longer, if such an event takes place and he will be able to get a reservation.

After finding some basic services, those found services must be refined and coordinated with each other. The proposed destination must be reachable by means of proposed transportation and the proposed events must take place during the holiday period. Those events have to be reachable in a comfortable way. The coordinated proposal may be shown to the customer now without checking availability. If desired by the customer, the system could also make alternative proposals. If the customer selects one of the proposals, it could happen, that no reservations could be made which would lead to some frustration.

Therefore, it would be better to first check for availabilities. One possibility to check for availability is to ask individual resource providers. We propose Web services to make such queries. However, this may lead to large search trees and a long processing time. Alternatively, we could store availabilities in a central database. This would contradict the idea of openness of the virtual enterprise and moreover, there would be a problem of consistent modeling of real availability. A solution often applied is to give a certain contingent of available rooms to a booking portal. In this case, however, the remaining resources cannot be used by the virtual enterprise.

A mechanism for improving the efficiency is a central data pool describing availabilities on an abstract level and a two step checking procedure. The system may determine that during a certain period in a certain region there will be no availability problem and the proposal will be shown to the customer immediately. If the customer has such requirements whose availability cannot be estimated easily, two sub-processes can be distinguished. The system can supply uncertain ranked offers in a short time. After customer's evaluation of these offers, the system may check the availability. The second way would be to check all availabilities first and to show only available solutions. This would also mean that the system checks solutions the customer is not interested in. This discussion shows that an intelligent reasoning dependent on season, customer preferences and other aspects must be designed for finding a user-friendly booking behavior.

If the user selects a booking solution, the system has to book several basic services. The customer's acceptance message can be interpreted as a contract with the virtual enterprise. When the virtual enterprise has booked the first resource, it is possible that the second resource has become unavailable. Since getting only one proposed service is not acceptable to the customer, the booking of the entire proposed package is defined as one transaction which either succeeds or fails. Therefore, the virtual enterprise will first reserve all resources and if this was successful, it will finally ask the customer for a definitive booking decision. This decision must be taken before a set deadline, derived from the deadlines of the single resource providers. If the customer accepts the final proposal, the booking will be processed.

We are developing a framework to support such dynamic planning tasks in virtual enterprises. We have used the actual version of the framework to implement a system to participate in the first competition on Web Service Discovery and Composition (Blake et al., 2005). In the next section, we describe the basic applied technology and short overview on work in the planning community. In the third section we present in detail the planning approach based on answer set programming and results of our system for the given benchmarks of the contest. Finally we conclude and stress the advantages of our approach.

BASICS AND RELATED WORK

In recent research related to Web services, several initiatives provide means to support integration of heterogeneous systems by defining languages, methods, or platforms. A Web service is a collection of protocols and standards used for exchanging data between applications. Software applications written in various programming languages and running on various platforms can use Web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.

The Web service Description Language WSDL (Booth and Liu, 2005) is a standard to describe Web services. It encompasses the definition of the messages and the message exchange protocol between a requester and provider agent as well as the definitions of operations and of bindings to a network protocol. The messages themselves are described abstractly and then bound to a concrete network protocol and message format. Web service definitions can be mapped to any implementation language, platform, object model, or messaging system. As long as both, sender and receiver, agree on the service description, (i.e. the WSDL document), the implementation behind the Web service can be any kind of software that is able to deliver the described service.

The definition of process languages such as BPEL (Andrews et al., 2003) or XPDL (WfMC, 2005) and the extension of Web service descriptions by OWL-S (Martin et al., 2004) aim towards static processes with a priori knowledge of all possible execution paths. In order to achieve dynamic processes to be composed during run-time, we need AI methods to compose services on-the-fly for an individual service requester.

Coping with Web services (or services in general) leads to the fundamental tasks of discovery, i.e., to find the appropriate services for a certain customer request, of composition, i.e., to bundle single or complex services to complex products, and of invocation and monitoring, i.e., to enact and supervise (complex) services. Additionally, with respect to composition, at least three other tasks are of interest:

- to specify goals for a composite,
- to specify constraints that have to be respected, and
- to provide means that help to analyze and to compare possible solutions against each other.

This work presents an approach that uses techniques from Answer Set Programming to solve some of the tasks required to build reliable and optimal compositions. In particular, we address the problem of finding and selecting an appropriate service, and the problem of limiting solution candidates to those that solve the goals and obey the restrictions imposed by the constraints. We demonstrate our work by using the challenges as well as the data format that has been specified for the Web Service Composition Contest (Blake et al., 2005).

The following WSDL document (taken from the composition contest) shows the data format and operation signature used for our work. It has a single operation with request/response protocol and simple data types - just a list of string properties.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="interopLab" xmlns="...>
<message name="findCloseHotel_Request">
  <part name="custStreetAddress" type="xsd:string"/>
  <part name="custCityAddress" type="xsd:string"/></message>
<message name="findCloseHotel_Response">
  <part name="hotelName" type="xsd:string"/>
```

```
  <part name="hotelStreetAddress" type="xsd:string"/></message>
<portType name="findCloseHotel"> <operation
   name="findCloseHotel">
    <input message="findCloseHotel_Request"/>
    <output message="findCloseHotel_Response"/>
  </operation> </portType> </definitions>
```

Since XML is verbose we use the following simplified textual notation to describe a Web service:

```
(findCloseHotel //operation
 (custStreetAddress custCityAddress ...) //request message
 (hotelName hotelStreetAddress ...)) //response message
```

Web Service Composition as Planning Problem

Dynamically composing complex products (or services) is a research topic in the AI world since a long time. The first well-known system was STRIPS the Stanford Research Institute Planning System that planned actions for a mobile robot in the research lab (Fikes and Nilsson, 1971). Theoretical investigations into this program have identified some drawbacks of the planner. First, there could be planning problems where the solution of one subgoal would destroy another subgoal. This led to the definition of the non-linear planning strategy. Moreover, the original algorithm was not able to handle parallel actions. Partial order planning or POP for short is a planning strategy that solves both problems (Weld, 1994). Instead of searching for a linear plan, required sequences between actions are identified first. Nevertheless, the original problem formulation with action that have pre- and post-conditions is still applied in modern planning systems.

So, not surprisingly, a survey of composition approaches for Web services conducted by Rao et al. (2004) revealed that typically AI methods from the planning domain are applied to the problem.

In general, a planning problem can be described as a five-tuple $\langle S, S_0, G, A, \Gamma \rangle$, where S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world, A is the set of actions the planner can perform, and the transition relation $\Gamma \subseteq S \times A \times S$ defines the precondition and effect for the execution of each action.

In terms of Web services, S_0 and G are the initial states and the goal states that are specified in the requirements of a service request. In the notation of the composition contest the initial state corresponds to a `provided` element and the goal state corresponds to the `resultant` elements of an XML document.

```
<Challenge><CompositionRoutine name="bookTrip">
<Provided>custStreetAddress,custCityAddress,... </Provided>
<Resultant>itineraryURL,...</Resultant>
</CompositionRoutine> ...</Challenge>
```

Again, a simplified notation helps to overcome the problem of verbose XML:

```
((problem bookTrip
  (provided custStreetAddress custCityAddress ...)
  (resultant itineraryURL ...)) ...)
```

Answer Set Programming with DLV

Answer set programming (ASP) is a form of declarative programming oriented towards difficult combinatorial search problems. It has been applied, for instance, to plan generation and product configuration problems in AI and to graph-theoretic problems arising in VLSI design. The original definition of answer sets for disjunctive logic programs was given by Gelfond and Lifschitz (1988, 1991).

Syntactically, ASP programs look like Prolog programs, but the computational mechanisms used in ASP are different: they are based on the ideas that have led to the development of fast satisfiability solvers for propositional logic. A main difference between Prolog and ASP programs is that ASP programs are strictly declarative, while Prolog programs have a procedural aspect. In Prolog, the order of rules as well as the order of subgoals in a rule matters while in ASP this makes no difference.

DLV (Datalog with Disjunction) is a powerful deductive database system (Leone et al., 2002). It is based on the declarative programming language Datalog, which is known for being a convenient tool for knowledge representation. With its disjunctive extensions, it is well suited for all kinds of non-monotonic reasoning, including diagnosis and planning. DLV was developed in cooperation between the University of Calabria and the Vienna University of Technology. More information about DLV is available at http://www.dbai.tuwien.ac.at/proj/dlv.

Datalog is a declarative (programming) language. This means that the programmer does not write a program that solves some problem but instead specifies what the solution should look like, and a Datalog inference engine (or Deductive Database System) tries to find the way to solve the problem and the solution itself. This is done with rules and facts. Facts are the input data, and rules can be used to derive more facts, and hopefully, the solution of the given problem. Disjunctive Datalog is an extension of Datalog in which the logical "or"-expression (the disjunction) is allowed to appear in the rules - this is not allowed in basic Datalog. A disjunctive Datalog program consists of an arbitrary number of facts, rules, and constraints. A rule takes the form:

head :- body

which can be read as "if the body is true, the head must also be true". The head can contain one or more disjunctive literals while the body can contain zero or more conjunctive literals. A fact is a special form of a rule in which the body is empty and thus is always true. The other special form of a rule, in which the head is empty, is called a constraint. The body of a constraint must not become true, because the head can never become true.

The Guess/Check/Optimize Methodology

The core language of DLV can be used to encode problems in a declarative fashion following a Guess/Check/Optimize (GCO) paradigm.

Given a set F_I of facts that specify an instance I of some problem **P**, a GCO program P for **P** consists of the following three main parts:

Guessing Part
The guessing part $G \subseteq P$ of the program defines the search space, such that answer sets of $G \cup F_I$ represent "solution candidates" for I.

Checking Part

The (optional) checking part $C \subseteq P$ of the program filters the solution candidates in such a way that the answer sets of $G \cup C \cup F_I$ represent the admissible solutions for the problem instance I.

Optimization Part

The (optional) optimization part $O \subseteq P$ of the program allows to express a quantitative cost evaluation of solutions by using weak constraints (Buccafurri et al., 2000). It implicitly defines an objective function $f: AS(G \cup C \cup F_I) \rightarrow \Re+$ mapping the answer sets of $G \cup C \cup F_I$ to (positive) real numbers. The semantics of $G \cup C \cup F_I \cup O$ optimizes f by filtering those answer sets having the minimum value.

In general, both G and C may be arbitrary collections of rules, and it depends on the complexity of the problem at hand which kind of rules are needed to realize these parts. In the next section some examples will illustrate the GCO paradigm.

SOLUTION

A Web service repository contains a set A of services. Each service maps to a WSDL operation with input and output parameters that represent preconditions and effect of this service. From the composition request two artificial services are created, named *start* (the service that provides the initial data) and *end* (the service that requires the goal data). The repository and the two additional services are depicted in Figure . The repository contains the services *a0-a9*. The arrows denote disjunctive ordering constraints, for instance, service *a5* can be scheduled after service *a2 or a3*.

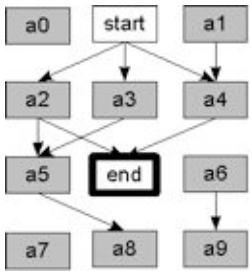

Figure 1: Services *a0-a9* in a repository, extended with artificial services *start*, *end*

In order to reduce the search space simple filtering is applied. Only those services are selected for further processing which may play a role in a solution:

Forward reduction. In the first step, breadth first forward search beginning at the *start* service is applied to reduce the set of services by such services that will in no case be invoked since their preconditions are never satisfied (cf. Figure 2).

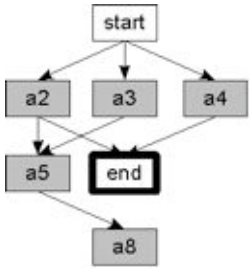

Figure 2: Services after forward reduction

Backward reduction. In the second step, using the (possibly) reduced set from step one, breadth first backward search starting from the *end* service is applied in order to remove services whose effects are never used. (cf. Figure 3).

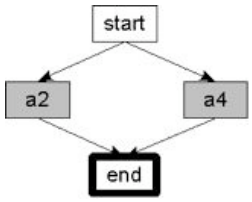

Figure 3: Services after backward reduction

Mapping Domain Model to DLV

The algorithm for mapping the services to the input of DLV:

For each service *a* from the set of services *A* (the repository) select the set *P* of services from *A* that have as an effect one of the preconditions *s* required by *a*. For each of the services in *P* one disjunctive entry in the body of the rule will be created. In other words, each service is represented as a formula in conjunctive normal form (CNF).

$$((a_0 \vee a_1 \vee \ldots \vee a_j)_1 \wedge (a_0 \vee a_1 \vee \ldots \vee a_j)_2 \wedge \ldots \wedge ((a_0 \vee a_1 \vee \ldots \vee a_j))_i)$$

with *j* being the services that are providers for the precondition *i*.

So for example, the requesting services in Figure 4 have the formulæ:

(1) $(a0 \wedge a0)$
(2) $((a0 \vee a1) \wedge (a2 \vee a3))$
(3) $((a0 \vee a1) \wedge (a1))$

Reduction for formulæ is as usual:

$$(x \wedge x) \rightarrow x \text{ and } (X \wedge Y) \rightarrow X \text{ if } X \subseteq Y.$$

In the example, the first formula becomes *(a0)*, the second remains unchanged, and the last is reduced to *(a1)*.

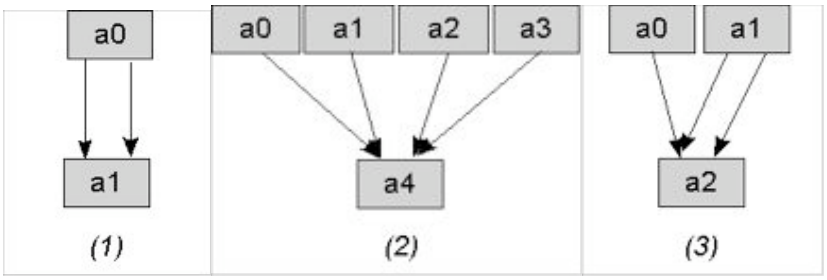

Figure 4: Reduction of preconditions

These formulæ are the input for the DLV solver. Each conjunction maps to one rule with the service name as body of the rule and the disjunctive part of the formula as the head of the rule. For instance, Figure 4 (*2)* yields two rules:

```
a0 v a1 :-a4.
```

```
a2 v a3 :-a4.
```

If the fact *a4* is present in the database, DLV computes four models, each being a solution (a plan) for the given problem:

```
{a4, a1, a3}
{a4, a0, a3}
{a4, a1, a2}
{a4, a0, a2}
```

Constraints

Constraints are means to reduce the result set either to answers that do not violate a (hard) constraint, or to limit the answer set to the cheapest answers. A constraint takes the form of a rule having no head. If the body becomes *true* the constraint is applied. The following example shows the layered structure of a program that follows the GCO paradigm. The guessing part consists of disjunctive rules that guess a solution candidate, the checking part consists of integrity constraints that check the admissibility of the candidate, and the optimization part consists of weak constraints.

```
a4. %fact
% disjunctive rules -> guess
a0 v a1 :-a4.
a2 v a3 :-a4.
% hard constraint -> check
:-a2,a0.
% weak constraints -> optimize
:~a0.[1:1] :~a1.[1.3:1] :~a2.[1:1] :~a3.[1:1]
```

This example shows weak constraints denoting the costs to invoke a service. Service *a1* has cost *1.3* while all other services have cost 1 associated. The second part of the cost array is used for the level the cost belongs to. The last line is a hard constraint that does not allow answers that have service *a0* as well as service *a2* in the result set. DLV comes up with a single answer in the result:

```
Best model: {a4, a0, a3}
Cost ([Weight:Level]): <[2:1]>
```

Plan construction for DLV input

The input for DLV is constructed by transforming the CNF representation for each service to rule(s). The literals in the head are the links from the providing service to the requesting service. Theses links express the ordering constraints between services. The body for each rule is the requesting service. The (artificial) *end* service denoting the goal is added as initial fact. This forms a graph with nodes *n(service)* representing services and links *l(from-service, to-service)* representing causal relationships. Searching is done backwards beginning at the goal, i.e. the *end* service.

Ordering constraints: Services can have cyclic dependencies between preconditions and effects, i.e. one service, e.g. service *a1* may have as a predecessor service *a0,* which in turn has *a1* as its predecessor. To avoid such answers (which are valid models but not valid plans) a constraint is applied that computes the path between nodes and becomes true if a node has been already visited.

A sample program:

Input: a problem *p1* and services *s0-s5* with preconditions and effects.

```
((problem p1(provided s)(resultant u v w)))
((s0(s)(a))
(s1(s)(b))
(s2(s)(c))
(s3(a)(u b))
(s4(b)(a c v))
(s5(c)(b w)))
```

A graphical representation of services and their dependencies is shown in Figure 5.

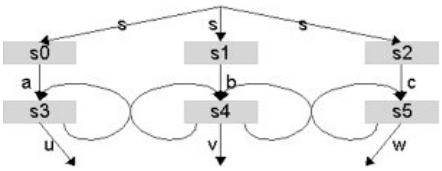

Figure 5: Example problem with cyclic dependencies

Transformation of this problem and services to the input format of DLV yields the following GCO program:

```
%the goal
n(end).
%service dependencies
l(s5,end)  :- n(end).
l(s4,end)  :- n(end).
l(s3,end)  :- n(end).
l(s0,s3)  v  l(s4,s3)  :- n(s3).
l(s5,s4)  v  l(s1,s4)  v  l(s3,s4)  :- n(s4).
l(s2,s5)  v  l(s4,s5)  :- n(s5).
%add node
n(X):-l(X,_).
%compute path
r(X,Y) :-l(X,Y).
r(X,Z) :-r(X,Y),l(Y,Z).
%check admissibility
:-r(X,Y),X==Y.
%cost
:~ n(s0).[1:1] :~ n(s1).[1:1] :~ n(s2).[1:1]
:~ n(s3).[1:1] :~ n(s4).[1:1] :~ n(s5).[1:1]
```

Description of the program:

- The first part is the representation of services and their dependencies, with one fact *n(end)*, i.e., the goal.
- The second part `n(X):-l(X,_)` adds a node for a selected link to the database (`X` is a variable and _ is an anonymous variable).
- The third part computes the path for any two nodes.
- The fourth part is a hard constraint. This constraint prohibits answers that contain a path from a node to itself from being included in the result set.

- The final part are the cost constraints, each service costs 1 unit.

The following code shows a single answer from the result set of the domain and problem above. It contains the selected services (as nodes `n(service)`), the ordering constraints (as links `l(from,to)`), and the cost for of the selected services.

```
{l(s5,end), l(s4,end), l(s3,end), n(s5), n(end), n(s4),
n(s3), l(s0,s3), l(s3,s4), l(s4,s5), n(s0)}
Cost ([Weight:Level]): <[4:1]>
... other answers (the problem has 8 answers,
3 are optimal in terms of costs)
```

From the eight solutions generated for the problem above two plans are shown in the following table: the cheapest answer (left) is constructed from four services, ordered sequentially and concurrently (requires 3 time steps). The most expensive answer (right) with eight services ordered in three parallel sequences (requires two time steps).

Table 1: Two solutions

Cheapest Plan	Fastest Plan
(1) s1	(1) s2
(2) s4	(1) s0
(3) s5	(1) s1
(3) s3	(2) s5
	(2) s4
	(2) s3

The following figure shows the resulting XPDL process for the first plan. The XPDL source code is given in the appendix of the paper.

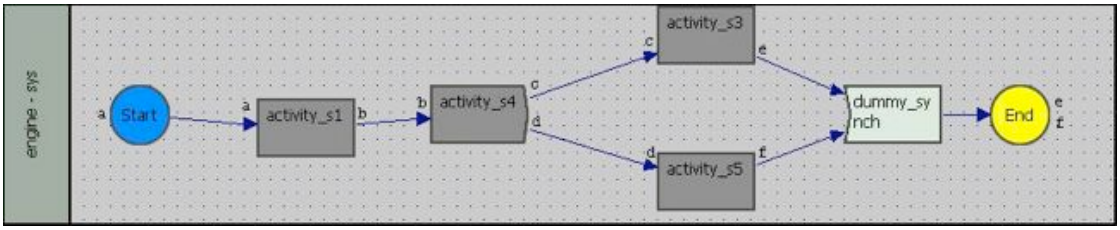

Figure 6: Visualized cheapest plan

Problem relaxation

Composition as described above results in totally ordered plans, i.e., every step is ordered with respect to every other step, and partially ordered plans, i.e., steps can be unordered with respect to each other.

Search space can become very large when many alternative services are available. So, for instance, when two services, `(s1(a)(b c))` and `(s2(b)(a d))`, and data `a b` are available and `c d` is a (sub)goal, then the answer will contain three compositions having the same costs:

```
s1 → s2, s2 → s1 and s1 && s2.
```

The symbols → and && denote sequence and concurrency, respectively.

The problem can be relaxed when search is done beginning at the start node, again using the GCO paradigm. The idea is that in the "guessing" part a service whose preconditions are satisfied is either selected (service is true) or not selected (service is false). When it is selected, the effects of the service are added to the facts which in turn may guess other services. The constraint is that an answer must contain the *end* node, i.e., the goal. Optimization is as usual with soft constraints.

```
%data available
a. b.
% Guess - either service is true or false
n(s2) v ~n(s2) :-b.
n(s1) v ~n(s1) :-a.
n(goal):-c,d.
% Check constraint - prunes illegal branches
:-not n(end).
% rules for adding effects of services
a :- n(s2). d :- n(s2).
b :- n(s1). c :- n(s1).
% Soft constraints, i.e. costs
:~n(s2).[1:1] :~n(s1).[1:1]
```

Running this program with DLV results in the single solution:

```
{a, b, n(s2), n(s1), d, c, n(end)}.
```

This solution has no ordering constraints, it contains just the selected services, the services that are not selected (none for this example) and the data elements added by selected services.

To gain an ordered solution we combine the two approaches: In a first step, guess and check is employed to find the cheapest solution candidates quickly. In the second step, starting backwards with the goal yields ordered plans from the unordered solution candidates as described above. Considering only services that are members of a solution found in the first step may reduce the search space extremely, but this may depend on the problem instance.

Performance Results

First tests with the data from the first Web service discovery and composition competition performed very well since the supplied data sets were rather small. Data sets for the second competition were larger and consequently in the temporal comparison to other programs drawbacks of our approach occurred and our program was ranked only third (see http://www.comp.hkbu.edu.hk/~ctr/wschallenge/).

We have tested our framework with the data provided by the Web Service Challenge syntactic repositories. The structure of these challenges is as follows: each repository contains several thousand WSDL files, each file has a single operation (i.e. the service) and a single portType element. A repository is associated with a single challenge file that has around ten challenges that refer to data elements from the WSDL descriptions. Each challenge can be solved with a fixed number of services in various defined combinations. For instance, a challenge named composition2-20-16-5 has as solution a sequence of seven services and each of the steps in the sequence can be performed by one of four services, resulting in 47=16384 possible solutions (since all services are distinctive). In order to reflect the fact that parsing WSDL documents is a substantial part of the overall composition time we have split the total composition time for a set

of challenges in two parts; one part for parsing the documents and constructing a memory representation and the other part that represents the time for composition of services.

The results show that parsing the WSDL files and construction of a repository in memory takes a lot of time for several thousand WSDL files. This is due to the fact that processing XML documents requires generally a lot of resources and also has to do with the Java implementation which is slower compared, for instance, with an C++ implementation.

The tests were performed on a PC with an Intel Pentium 2GHz processor with 1 Gigabyte Memory and Windows 2000 operating system.

Table 2: Results for the benchmarks

Instance	WSDL docs	Challenges	Parse time (sec)	Composition time (sec)
composition1-20-4	2156	11	13.98	1.59
composition1-20-16	2156	11	15.93	1.73
composition1-20-32	2156	11	24.84	2.02
composition1-50-4	2656	11	24.86	1.86
composition1-50-16	2656	11	27.36	1.98
composition1-50-32	2656	11	29.40	2.12
composition1-100-4	4156	11	44.40	2.31
composition1-100-16	4156	11	40.00	2.65
composition1-100-32	4156	11	52.10	2.76
composition2-20-4	3356	11	33.00	134.20
composition2-20-16	6712	11	91.30	136.30
composition2-20-32	3356	11	38.70	135.60
composition2-50-4	5356	11	61.81	135.70
composition2-50-16	5356	11	66.03	138.90
composition2-50-32	5356	11	71.25	136.60
composition2-100-4	8356	11	125.45	138.70
composition2-100-16	8356	11	130.00	138.00
composition2-100-32	8356	11	102.00	139.00

The measurements show clearly that our program has problems with large Web service repositories where the compositions have many solutions. One reason is the time and space required to parse these with standard Java libraries. Another reason is of course our attempt to produce all solutions.

We believe, however, that it is not realistic that there exist so many possible solutions that are also even good.

Implementation Details

We have implemented the composition within the MOVE framework. MOVE is a framework for designing and execution of processes in virtual enterprises. The framework is based on Java, open-source components and the Eclipse development environment. The main focus of the MOVE framework is laid on Web service integration. Processes are modeled graphically and transformed into XPDL. Such XPDL processes are executed with a workflow engine whereas each activity in an XPDL process may represent a Web service call. Moreover, there are specialized Web services called infrastructure Web services. Infrastructure Web services provide

frequently used services in a Web service integration framework. Such common infrastructure Web services include for example the transformation of XML documents and the discovery of Web services at process run-time. The composition of Web services from discovered Web services also represents an infrastructure Web service and was implemented in this manner.

The composition Web service consists of several components shown in Figure 7. These components have fixed interfaces and can be substituted at run-time in a flexible way when required. Thus, Web services can be discovered at different sources, different solvers can create different compositions, and different transformers can transform results to different data formats.

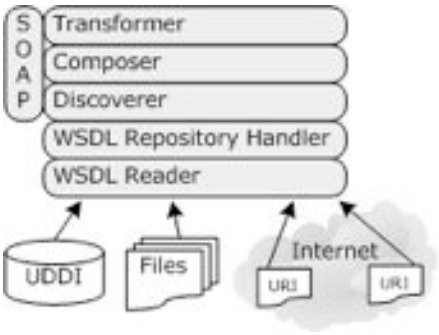

Figure 7: Architecture

The WSDL reader component processes WSDL documents and converts them into a Java data structure. Depending on the WSDL reader's implementation, any source of WSDL document is possible: Files on a local file system, documents on remote servers, documents in an XML database or in some registry, and so on. How WSDL information is transformed to a Java data structure is arbitrary, too. Our implemented WSDL reader uses WSDL4J to transfer WSDL information to a Java data structure and is able to process WSDL documents residing in the local file system or in the Internet via http protocol. WSDL4J is a library that complies with the JWSDL standard. JWSDL provides a standard for Java APIs that can be used for representing, manipulating, reading and writing WSDL. Future implementations of the WSDL reader may consider registry sources for WSDL documents like UDDI and may use specialized SAX parsers instead of the extensive WSDL4J library for performance reasons.

To provide unified, comfortable and flexible access to a set of Java WSDL data structures, the WSDL repository component is used. Based on certain environment variables like the number or sources of WSDL documents, different performance enhancement strategies may be implemented. Our implementation provides solutions for different numbers of WSDL documents. For a small number, the WSDL repository handler loads all WSDL documents at program start a priori. For a large number of WSDL documents that is likely to change, another implementation lazy-loads WSDL documents just before they are needed. Also caching strategies may be considered in future implementations. When WSDL sources are prone to provide slow and unreliable access like it is the case with WSDL documents in the Internet, caching will provide a significant performance and reliability improvement.

The Discoverer component provides searching capabilities for discovering Web services that fulfill certain requirements. Such requirements could be based on the currently available set of input data and the expected output data. Web services whose needed input data does not exceed the available input data set and whose output satisfies the expected output data could be searched and discovered. In addition, semantic Web service concepts could be implemented here to enable semantic search and discovery of suitable Web services.

The Composer component is responsible for building complex services. Apart from applying the DLV solver other planning and optimization software may be used.

The Transformer component can be used to manifest service compositions into different representations. Translating service composites to an executable software like a Java algorithm or a Web service orchestration such as XPDL or BPEL(4WS) are options as well as creating DLV input files from WSDL documents. Furthermore, visual representations of the composition using e.g. SVG may be worthwhile manifestations.

Not only the Composer component, but also the Discoverer and Transformer may represent reasonable infrastructure services. Thus, as shown in Figure 7, these components publish their functionality via SOAP interfaces also and can be used in the MOVE framework as infrastructure services to discover services or to transform compositions.

SUMMARY AND OUTLOOK

We have presented briefly a general framework for developing virtual enterprises and in detail an approach to construct business processes from basic Web services using AI-techniques. Service descriptions given in WSDL are mapped into a rule-based language that allows to search a repository efficiently and to build solutions that solve a goal with respect to soft and hard constraints. In contrast to other solutions, we generate all possible solutions for a given problem.

In our contribution to the Web service composition contest (Dorn et al., 2004) we have shown the usefulness of our approach. The strength of our approach is that it provides all solutions for a given problem despite the costs in computation time and space. Our tool has performed best in finding solutions for a given set of problems in the first competition. The tool was also used to analyze the randomly generated data for the second composition contest in Beijing in October 2005. Finding all possible solutions helps to control the complexity of the challenges. Additionally, having all solutions computed ensures that solution claims from competitors can be verified. In the second contest, our program was ranked only third. One reason for this is that the required time for finding a solution was the selective criteria. Since our tool is implemented in Java and uses Java libraries for standard functions the speed is restricted in contrast to other programming languages.

Nevertheless, our solution exhibits some strong benefits that shall be stressed here. Due to the nonlinear planning strategy our solution detects also which Web services may be executed in parallel. This would result in a shorter execution of the whole set of Web services. Moreover, we support a soft evaluation of solutions. This means we can attribute costs to single Web services, aggregate them for a solution and rank the found solutions by means of a cost function. Finally, our solution is part of larger framework. The found solution can be mapped into a process language such as XPDL and executed in the framework. Generic solutions could also be deployed as a new Web service. In this case no planning has to be performed again.

In related work, other methods from the planning domain are applied. For instance, Peer (2004) maps WSDL documents to PDDL (McDermott 2000) a language defined for planning competitions. A planner may then process the resulting PDDL statements. This is similar to our approach. However, a number of researchers see plain WSDL as too limited to express preconditions and effects of services and they propose a richer syntax and semantic for service descriptions. As an example, Sirin and Parsia (2004) propose to annotate Web services with semantic information expressed in OWL-S. They show how a reasoner for description logics (DL) in combination with a planner can be applied to compose services. However, they claim that using DL may cause performance problems.

The current solution is performing very well in a rather simple domain. We see a number of possible extensions for our framework to cope with "real world" requirements. First of all, the message format is very simple and may be enriched with complex data types or ontological information, for instance, OWL (Bechhofer et al., 2004) expressions. We think that a logic reasoner like DLV fits very well the needs imposed by Description Logics. Second, services in a repository may have a control flow already defined and the challenge is to generate solutions that combine different processes. Again, we think that such requirements can be solved efficiently within our framework, but this needs more clarification. Third, using a simple cost function such as plainly adding service costs may be not suitable to distinguish solution candidates and may be replaced by a complex utility function that represents better the requirements of a single service request. Finally, the contest assumption that all Web services are already known is not a realistic point of view.

ACKNOWLEGMENT

This reported work is performed in the MOVE[1] project supported by ec3, the E-Commerce Competence Center in Vienna, its industrial partners as well as the Austrian Government.

The authors would like to thank also the organizers of the Web service contests M.B. Blake, W.K. Cheung, K.C. Tsui, and A. Wombacher for providing services and problem instances as input data. This helped to verify the correctness of the results of the composition tool as well as insight in requirements for our framework.

REFERENCES

Andrews, T., et al.: (2003) Business Process Execution Language for Web Services. 2nd public draft release, Version 1.1

Bechhofer, S., et al. (2004) OWL Web Ontology Language Reference.http://www.w3.org/TR/owl-ref/.

Blake, M.B., Tsui, K.C., Wombacher, A. . (2005) The eee-05 challenge: A new Web service discovery and composition competition. In: EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05), Washington, DC, USA, IEEE Computer Society 780-781

Booth, D., Liu, C.K. (2005) Web Services Description Language (WSDL) Version 2.0 http://www.w3.org/2002/ws/desc/.

Buccafurri, F., Leone, N., Rullo, P. (2000) Enhancing disjunctive datalog by constraints. Knowledge and Data Engineering 12, 845-860

Dorn, J., Hrastnik, P., Rainer, A. (2005) Web service discovery and composition with move. In: EEE'05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05), Washington, DC, USA, IEEE Computer Society 791-792

Gelfond, M., Lifschitz, V. (1988) The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K., eds.: Proceedings of the Fifth International Conference on Logic Programming, Cambridge, Massachusetts, The MIT Press 1070-1080.

Fikes, R. E. and Nilsson, N. (1971).STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189-208.

Gelfond, M., Lifschitz, V. (1991) Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365-386

[1] The acronym MOVE stands for Management and Optimization of business processes in Virtual Enterprises

Hrastnik, P. (2004) Execution of Business Processes Based on Web Services, Int. J. Electronic Business, Vol. 2, No. 3.
Leone, N., Pfeifer, G., Faber, W., Calimeri, F., Dell'Armi, T., Eiter, T., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., Perri, S., Polleres, A. (2002) The dlv system. In: JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence, London, UK, Springer-Verlag 537-540
Martin, D., et al. (2004) OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1.
McDermott, D. (2000) The 1998 AI Planning Systems Competition. AI Magazine 2 (2), 35-55
Peer, J. (2004) A pddl based tool for automatic Web service composition. In Ohlbach, H.J., Schaffert, S., eds.: PPSWR. Volume 3208 of Lecture Notes in Computer Science, Springer 149-163
Rainer, A. (2004) Web-Centric Business Process Modelling, Int. J. Electronic Business, Vol. 2, No. 3.
Rao, J., Su, X. (2004) A survey of automated Web service composition methods. In Cardoso, J., Sheth, A.P., eds.: SWSWPC. Volume 3387 of Lecture Notes in Computer Science., Springer43-54
Sirin, E., Parsia, B. (2004) Planning for semantic Web services. In: Semantic Web Services Workshop at 3rd International Semantic Web Conference (ISWC2004).
W3C. (2003) Simple Object Access Protocol (SOAP) 1.2, http://www.w3c.org/TR/2003.
Weld, D.S. (1994).An Introduction to Least-Commitment Planning, AI Magazine, 15(4) 27-61.
WfMC (2005) XML Process Definition Language (XPDL) 1.09, http://www.wfmc.org.

ABOUT THE AUTHORS

Biographical notes of each author

J. Dorn studied computer science and economics at Technische Universität Berlin. He received his Ph. D. from Technische Universität Berlin in 1989 for a thesis on knowledge-based reactive robot planning. From 1989 to 1996, he was head of a group at Christian Doppler Laboratory for Expert Systems in Vienna that has developed several scheduling expert systems for the Austrian steel industry. He is now professor of business information systems at Vienna University of Technology and leads the Move project group at ec3.

A. Rainer studied business computer science at Vienna University of Technology. He is a member of the scientific staff at ec3 since 2001 and has finished in 2006 his Ph. D. on Process verification and Web Service Composition.

P. Hrastnik studied business computer science and economics at University of Vienna. After receiving the degree, he was employed by an Austrian mobile telecommunications provider. Since 2001, he's been at ec3. His Ph. D. to be finished in 2006 focuses on integration of transaction models in Web processes based on Web services. Peter was member of the second Web service composition contest in Bejing.