Automated Web Service Composition Using Classical Represenation of Planning Domain

Abstract

The ability to automatically answer a request, that requires the composition of a set of web Services, has received much interest as it support B2B(Business to Business) applications. Most existing Web Services composition researches consider that the request specifies the subset of Web Services and the order between them. This order depands on the result of executing a web service, and the role of the proposed systems is to find a Web Services to execute the request. In our approach we overcome this limitation in order to answer a request that specifies only the initial and the final state, and the system selects and orders the set of web services automatically to answer a request by using the Artificial Intelligence techniques.

INTRODUCTION

Web Services are distributed software components that can be exposed and invoked over the internet using standard protocols. They communicate with their clients and with other Web Services by sending XML based messages over the internet (Peer March 22 2005). They are self-contained, self described, active and modular software applications that can be advertised, discovered and invoked over the internet (Gottscalk & Team. 2000). For example, an airline travel service or a book-buying service (amazon.com). Many languages have been developed -in industrial and academic communities- to specify Web Services, such as BPEL4WS(Business Process Execution Language for Web Services) (Andrews & Weeravarana 2003),WSDL (Web Services Description Language) (Roberto Chinnici 2004).

The work on web service composition in the university of Trento presented in (Pistore *et al.* 2004) define the planning domain by a tuple $D = \langle S, A, I, T \rangle$ where S is the set of state, A is the set of actions, $I \subset S$ is th set of initial state, and $T : S \times A \rightarrow 2^{\mathbf{S}}$ is the transition function. The planning problem is to find a set of actions to reach a defined goal with some constraints on the execution of the plan. They use the Model Based Planner MBP (Bertoli *et al.* 2001) based on model checking (Pistore & Traverso 2001) techniques to find the solutions to their problems. The inconvenience of this models is that we must define all the domain states, and the transition actions from all states and actions before planning for any request. Another inconvenience is that when we want to add a new action(or service) to the domain we must redefine the domain states and the transition functions again.

In his thesis (Lazovik, Aiello, & Papazoglou 2003), Alexander Lazovik defines the problem of service composition by the same way defined obove, but with adding the properties to interacting with the client in the execution phase based on the principle of interleaving the planning and the execution phases. His planner is based on the constraint satisfaction technique, his choice is motivated by the fact that in a web service scenario, users may wish to know why certain solutions are preffered to others.

In (McIlraith & Son 2002), a method is presented to compose web services by applying logical inferencing techniques on pre-defined plan templates. The service capabilities are described in Prolog, and given a goal description, the logic programming language of GOLOG is used to intantiate the appropriate plan for composing the Web Services. Golog is implemented in Prolog and based on the situation calculus and it supports specification and execution of complex actions in dynamic systems. The inconvenience of this approach is that the plan is pre-defined!.

Our model will overcome the weakness of the first two approaches defined above to permit the domain to be dynamic i.e. we can add new services to the domain without changing any other predefined part of the domain, and we exceed the weakness of the third approach by defining the request not by a predifined-plan, but only with specifiying the inital and the final states (what the client has as data, and what he wants as result).

Artificial Intelligence (AI) planning techniques can contribute in solving this kind of application(Yan 2006), in fact services can be modeled as actions and the business process as planning to connect the Web Services. The main contribution of this paper is to build the plan automatically, by sensing the world to explore the available services, and by developing from the initial state the different intermediate states, by appliying services, until the stated goal and returning the plan (set of services to reach goal), or returning failure.

The rest of this paper is organised as follows: in section

we present the motivated example of our works, in section we present the formal framework of our model by defining the services, plan, and request. Then in section we present the planning algorithms (Tree-search and Graph-Plan) to find solutions to our problem, and finally we conclude by analysing the results of implementation, and describing our future work.

EXAMPLE

Suppose a set of Web Services are dedicated to dealing with files, images and tracks.

- The first Web services translate files, and has three services:
  1. `fr2en`
     tranlates file from French to English.
  2. `en2ar`
     tranlates file from English to Arabic.
  3. `en2fr`
     tranlates file from English to French.
- The second Web Services transforms files, it has two services:
  1. `latex2doc`
     transforms file from latex to doc format.
  2. `doc2pdf`
     transforms file from doc to pdf format.
- The third Web Services merge files, it has two services:
  1. `mergepdf`
     merges two pdf files in a third file.
  2. `mergedoc`
     merges two doc files in a third file.
- The forth Web Services transform the format of an image, it has two services:
  1. `PNG2JIF`
     transforms an image from PNG to JIF format.
  2. `JPEG2PNG`
     transforms an image from JPEG to PNG format.
- The fifth Web Services tranform the type of a track, it has two services:
  1. `mp3towav`
     transforms a track from mp3 to wav format.
  2. `rm2mp3`
     transforms a track from rm to mp3 format.

As an example of a problem to solve, suppose that we have two files:the first is a doc format written in English, the second is in latex format written in French. The goal is to have a pdf file that contains the content of two files translated to Arabic.

FORMAL FRAMEWORK

In our formal framework based on Planning-Graph techniques (Malik ghallab Jun 2005), The main difference between this approach and our model, is that we need to express that our domain doesn't have a fixed object in all states, and the execution of actions will change the properties of objects and relations between them, but in the domain of Web Services the execution of services can eliminate and create variables which define the domain.

Preliminairies and Definitions

The domain $\mathbf{D} = (C, F)$ is defined by a set of Web Services $\mathbf{C} = (WS_1, WS_2, ..., WS_n)$ that we call a community of Web Services, and a set of predicates **F=** $\{F_1, F_2, ..., F_n\}$ to specify the preconditions and effects, of executing services on the world state.

In our example , the community of Web Services is defined by

C=(WS_1,WS_2,WS_3,WS_4,WS_5)

, and the set of predicates is defined by

F= {(**ar** f),(**fr** f),(**en** f),(**pdf** f),(**doc** f),(**merge** f1 f2 f),(**JPEG** i),(**mp3** t),...} where f, i, and t are respectively file, image, and track.

definition 1 *Web Services: A Web Service WS_i is defined by* WS_i = (T_i, A_i, S_i)

with

- T_i : is the type of service.
- A_i : is the set of Web Services attributes.
- $S_i = (s_{i1}, ..., s_{im})$ is the set of services composing WS_i.

Example: In our Example, WS_1 = ({translation},{IP='127.127.0.41'},{fr2en, en2ar, en2fr})

definition 2 *State: s=(V,P)*

A state s of the domain is defined by a set of variables V and their types, and a set of predicates P specifiying the properties of these variables and the relationship between these variables. A specification for the initial state in the example is: s=({(F1: **file**),(F2: **file**)},{(**doc** F1),(**en** F1),(**latex** F2),(**fr** F2)}).

definition 3 *Service : A service S_i in Web Services WS_i is defined by* $s_{ik} = (Pin_{ik}, Pout_{ik}, Pinout_{ik}, Effet, Prec)$ *where:*

- $Pin_{ik} = \{pin_{ik1}, pin_{ik2}, ...\}$: Input parameters of service (an image for example).
- $Pout_{ik} = \{pout_{ik1}, pout_{ik2}, ...\}$: Output parameters of service.
- $Pinout_{ik} = \{pinout_{ik1}, pinout_{ik2}, ...\}$: Input-output parameters of service.
- Prec: Conditions that have to be true for all the **in** and **inout** parameters P_{in} et P_{inout} of S_i.
- Effets: The effects of executing a service to the current state of domain.

We notify that we have two kinds of effects for the execution of service, implicit and explicit effects, the implicit effects consist of eliminating all the input variables, and creating new variables for each output parmeter. The explicit effects consist of eliminating some predicates $Effet^-(s_{ik})$, and creating a set of predicates $Effet^+(s_{ik})$.

$$Effet(s_{ik}) = Effet^+(s_{ik}) \cup Effet^-(s_{ik})$$

Example: in Example , For the service **mergepdf** of the Web Services WS_3:

- Pin= { (f1:file),(f2:file) }.
- Pout= {(f file) }.
- Pinout= { }.
- Prec={(**pdf** f1),(**pdf** f2)}.
- $Effet^-$={(**pdf** f1),(**pdf** f2)}.
- $Effet^+$={(**pdf** f)}.

definition 4 *Plan: A plan is defined by a sequence of set of services* $\Pi =< \pi_1, \pi_2, ..., \pi_n >$*, with* $\pi_i = (s_{i1}(v1, ..., Vm), ..., s_{in}(v1, ..., Vl))$ *is a partial plan.*

The input parameters of a service s_{ik} of the partial plan π_i depend on one or multiple output parameters of the partial plan π_{i-1}. The order of execution of the plan Π is to execute all the services of the partial plan π_i in any order or even in parallel, then execute the services of π_{i+1} and so on.

Example: A solution of defined problem in is $\Pi =< \pi_1, \pi_2, \pi_3, \pi_4, \pi_5 >$, with:

- π_1=(fr2an [F1], an2ar [F2]).
- π_2=(an2ar [F1], doc2pdf [F2]).
- π_3=(latex2doc [F1])
- π_4=(doc2pdf [F1]).
- π_5=([#0file]=mergepdf[F1, F2]).

definition 5 *Executable service: a service s is executable in a state e of the domain if and only if :*

- For every parameter $in \in \{P_{in} \bigcup P_{inout}\}$ there exists a variable of the current state of the world having the same types.
- The set of preconditions on input and inout Parameters are all satisfaisable.

Example: In the initial state of the example , doc2pdf[F1],en2ar[F1],latex2doc[F2] and fr2en are executables services.

After the execution of services, the state of the world change , new variables are generated (those are the output parameters of the executed service) and a new predicate will be attributed to the set of generated variables , Further updates will be held for all elements P_{inout} based on the effects of service.

definition 6 *Request formalisation : A request R=(**D**,e_0,g) is defined for a domain **D** for the initial state e_0 and the goal g.The initial and final state are defined by a set of variables with a set of associated predicates.*

Example: In the example , e_0=[{(F_1:file),(F_2:file)},{(doc F_1),(en F_1),(latex F_2),(fr F_2)}].

and g=[{($\#F_0$:file)},{(pdf $\#F_0$),(ar $\#F_0$),(merge F_1 F_2 $\#F_0$)}].

The aim of using the symbol # before the name of variable, is to announce that any other variables beginning with # can replace it in the domain.

Planning Algorithm

Tree-search

We have two methods to construct the solution of our problem, the first is the classical solution, the Tree-search algorithm 1. The strategy is depth or width search. By this method we obtain a sequential plan of services. We must tranform this plan to a sequetial of set of partial plan (set of independant service) $\Pi =< \pi_1, \pi_2, ..., \pi_n >$. The algorithm 2 contains the implementation of this algorithm.

The general idea of the algorithm 2 is as follows: We route the plan of the sequence of services, service by service, then we build during our route the sequential overall state, the plan sequence is composed of many levels, each level is built on a set service which are all independent between them, and each of these services depend on a service in the $level_{i-1}$.

Algorithm 1: Tree-search algorithm

```
Tree-search(domain D,state e_0,state g):(plan p or
failure)
initialisation of the search tree using the initial state of
problem; selection of the list of services from the
domain D;
found=false;
while not found do
    if there are no executable service for expansion of
    the tree then
        return failure
    end
    choose a leaf node for expansion according to
    strategy
    if node contains g then
        return list of services for reaching node
    else
        expand the node with all executable services
        and add the resulting node to the search tree.
    end
end
```

This Algorithms 2 takes as input the plan of sequential service, and as a result it makes a plan of sequential set of service.

The function newLevel, creates a new level (partial plan) π and adds it in the end of the plan Π.

The function level(Π,i) return the partiel plan π_i of Π.

the function $depend(service, \pi)$ tests if one of the **input** or **inout** parmeters of service, uses the **inout** or **output** parameters of one of the services constructing π.

The function addService(service,Π,j) adds service to the π_j

of the plan Π.

Algorithm 2: Plan tranformation algorithm

PlanTransformation(Π_{in}):(Π_{out})
n $\longleftarrow$ size(Π_{in})
currentService $\longleftarrow$ sercvice(1,Π_{in})
newLevel(currentService,Π_{out})
i $\longleftarrow$ 2
while $i < n$ **do**
  currentService $\longleftarrow$ service(i,Π_{in})
  π_{final} $\longleftarrow$ level(Π_{out},size(Π_{out}))
  if *depend(currentService,*π_{final}*)*
  then
    newLevel(Π_{out},currentService)
  else
    j $\longleftarrow$ size(Π_{out})-1
    while $j > 0$ **do**
      $\pi_{courant}$ $\longleftarrow$ level(Π_{out},j)
      if *depend(currentService,*$\pi_{courant}$*)*
      then addService(currentService,Π_{out},j+1)
      j $\longleftarrow$ 0
    end
    i $\longleftarrow$ i+1
    if *j=0* **then** addService(currentService,Π_{out},1)
  end
end
return Π_{out}

Graph plan method

The *planning graph techniques* introduce a very powerful search space called a planning graph. It starts from the state space planners who provide a plan as a sequence of actions (or services), and from the plan space planners who synthetize a plan as a partially ordered set of services(Malik ghallab Jun 2005). A major contribution of the Graphplan planner is a relaxation of the reachibility analysis. This approach provides an incomplete condition of the reachibility through a planning graph. However, this is not a sufficient condition anymore. This weak reachbility condition is compensated by a low complexity: the planning graph is of polynomial size, and can be built in polynomial time in the size of the input.

The basic idea in a planning graph is to consider at every level of this structure not individual states but, to a first approximation, the union of sets of propositions in several states. A planning graph is a directed layered graph: arcs are permitted only from one layer to the next. Nodes in level 0 of the graph corresponds to the set P_0 of predicates denoting the initial state s_0 of a planning problem. Level 1 contains two layers: an action level A_1 and a predicate level P_1, and so on for the other levels .

Before continuing, we need to define the *mutex* for services and actions.

definition 7 *Independant services: Two services s_1 and $S2$ are independant if and only if:*

- $\{Pinout_1 \cup Pout_1\} \cap \{Pin_2 \cup Pinout_2\} = \oslash$.
- $\{Pinout_2 \cup Pout_2\} \cap \{Pin_1 \cup Pinout_1\} = \oslash$.

definition 8 *Mutex: Two services s_1 and $s2$ are mutex if either s_1 and s_2 are dependant, or if a precondition of s_1 is mutex with a precondition of s_2. Two preconditions p and q in P_i are mutex if every service in A_i that has p as a positive effect is mutex with every action that produces q, and there is no action in A_i that produces p and q.*

The GraphPlan algorithm performs a procedure close to iterative deeping, discovering a new part of the search space at each iteration. It iteratively expands the planning graph by one level, then it searches backward from the last level of this graph for a solution. The first expansion, however, proceeds to a level P_i in which all the goal propositions are included and no pairs of them are mutex() and the set of services executed for reaching g are mutex and so on until reaching P_0, and then the plan is found , or until reaching failure($P_i = P_{i+1}$ and no plan is found). In figure 1 we give the execution of Graph-plan, to the example , but with changing in the initial state to obtain a readable figure (Dashed lines correspond to negative effects, and not all arcs are shown) .

Example The goal g=[{(#F_0:file)},{(pdf #F_0),(ar #F_0),(merge F_1 F_2 #F_0),(ar #F_1),(ar #F_2)}]. of example is in P3 without mutex, the only actions in A3 acheving g is $\pi_3 = [mergepdf]$.
At level 2, the new goal g1= {goal-effects(π_3)+preconditions(π_3)}, and the new goal become: $\{(pdfF1)(arF1)(pdfF2)(arF2)\}$, $(arF1)$ is achieved by en2ar and $(pdfF2)$ is achieved by doc2pdf , and $\pi_2 = [en2ar, doc2pdf]$.

At level 1, the new goal is g2= {goal-effects(π_2)+preconditions(π_2)}, and the new goal become: $\{(pdfF1)(enF1)(pdfF2)(arF2)\}$, $(enF1)$ is achieved by fr2en, and $(arF2)$ is achieved by en2ar.

and level 0 is successfully reached.

Algorithm Implementation and Results

We implemented the tree-search and the GraphPlan algorithm.

In our model we define or use a planning domain definition language to specify the set of availables Web Services, the set of predicates, the set of variables and parameters types of the domain. In our implementation we use a part of the PDDL language, and extend it to adapting our model. PDDL(M.Ghallab 1998) is intended to express the physics of a domain, that is, what predicates there are, what actions(or services) are possible, what the structure of compound actions is, and what the effects of actions are. The hope of developing this standard language is to encourage empirical evaluation of planner performance, and development of a standard set of problems all in comparable notations. In general the majority of planners will not handle the entire PDDL language,and will extend the notation.

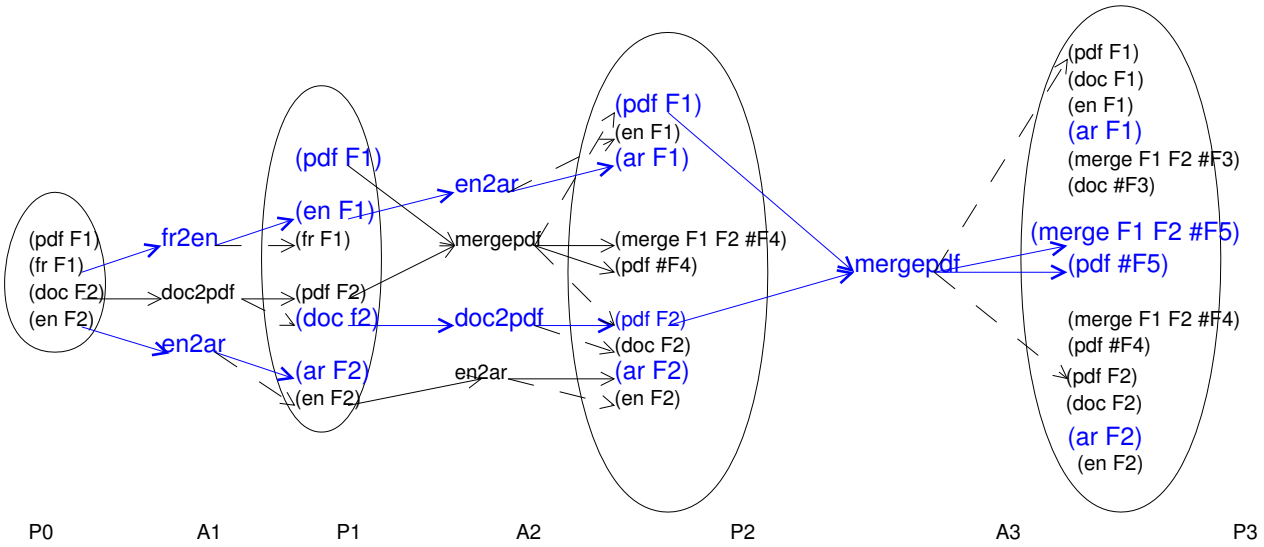

Figure 1: Graph plan execution

Example of service specification Here we give the definition of the merge service s_{12} (called en2fr-english to french) defined in section .

```
(
:action en2fr
:in\_parameters ()
:in\_out\_parameters (?f4 -file)
:out\_parameters ()
:precondition (en ?f4)
:effect (fr ?f4)
)
```

and the defintion of the merge service s_{31}(called mergepdf) defined in section

```
(
:action mergepdf
:in\_parameters (?f5 -file ?f6 -file)
:in\_out\_parameters ()
:out\_parameters (?f7 -file)
:precondition (pdf ?f5)*(pdf ?f6)
:effect (pdf ?f7)*(merge ?f5 ?f6 ?f7)

)
```

Planning problem and result Now we will give the specification of the initial and the goal of the problem of planning presented in .

```
init=
(
parameters=
(F1 file)*(F2 file)

predicates=
(fr F1)*(latex F1)*(doc F2)*(an F2)
)
goal=
(
parameters=
(\#F0 file)

predicates=
(ar F1)*(ar F2)*(pdf \#F0)*(merge F2 F1 \#F0)
)
```

Tree-search implementation

The result of the execution of our algorithms is:

- **in depth:** Π=
  - π_0=(fr2an [F1], an2ar [F2])
  - π_1=(an2ar [F1], doc2pdf [F2])
  - π_2=(latex2doc [F1])
  - π_3=(doc2pdf [F1])
  - π_4=([#0file]=mergepdf[F2, F1])
- **in width:** Π =
  - π_0=(latex2doc [F1], an2ar [F2]).
  - π_1=(fr2an [F1]),(an2ar [F1]).
  - π_2=([#27 file]=mergedoc [F2, F1]).
  - π_3(doc2pdf [#27]).

Note: All the variables beginnning by the # character are generated variables (result for executing a service that has output parmeters), this means that there name isn't important and that anyone from them can replace the other. In the result in wedth (#0 play the role of #F0).

Empirical result We tested our algorithms for 7 examples (the specification is given in the Appendix) that contain many variables and many types of variables, we have 16 services in the domain, and the result are in Figure 2 and Figure 3. From the two diagrams we can see that the depth strategy is very effective and in a few seconds we obtain a plan by

expanding a few number of nodes, but the width methods obtains for the same problems the results by expanding a huge number of nodes that require a few minutes, and for P4 problem more than 2750 nodes without obtaining a result. Another remark can be dedicated, the order of expanding the services in a state is very important for decreasing the number of developed nodes in width strategy.

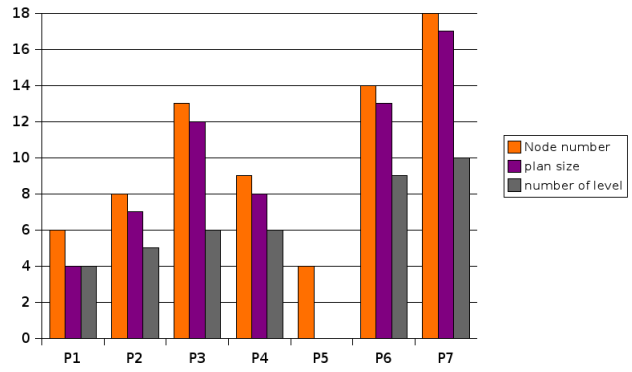

Figure 2: depth result

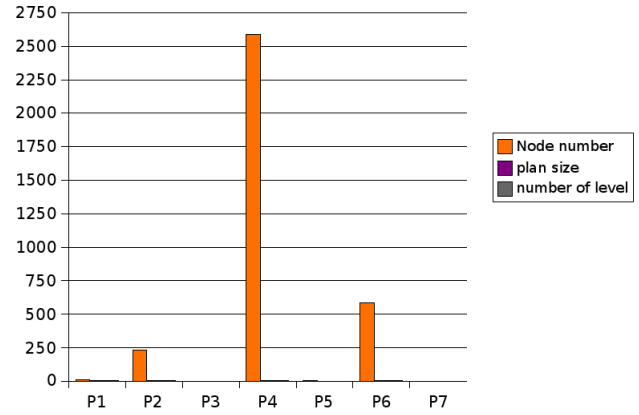

Figure 3: width result

Graph Plan implementation

By using the GraphPlan algorithm, we obtain solutions for simple problems , but not for complex (problems that contain more than 3 variables), because we obtained combinatorial explosion, the combinatorial explosion is due to the computation of all possible combinations at every level of expanding the planning graph. In fact, we must calculate every possible combination, of all the variables that correspond to each inputs of all services. furthermore, for testing if we reached a level that contains the goal state, and since every state parameters (parameters that begins #) of a data types can play the role of other parameters of the same types, we should replace every parameter in the set of predicates, by one of the parameters of the last level predicates.

Here we give the specification of the problem illustrated in 1

```
init=
(
parameters=
(F1 file)*(F2 file)

predicates=
(fr F1)*(pdf F1)*(doc F2)*(en F2)
)
goal=
(
parameters=
(\#F0 file)

predicates=
(ar F1)*(ar F2)*(pdf \#F0)*(merge F2 F1 \#F0)
)
```

And we obtain fastly this solution:
by Graph-plan: Π=

- π_0=(fr2en[F1], doc2pdf[F2])
- π_1=(en2ar[F1] en2ar[F2])
- π_2=([#0 file]=fusionpdf[F2, F1])

Conclusion And Perspective

In this article we give a new view to the problem of composition of Web Service, by modeling the problems as a planning problem, and by using the artificial intelligence technique. In our work we propose a model to exceed the limitations of other approaches, by giving a dynamic and distributed definition of our domain, this allows us to (add, remove, replace) services without recalculating another part of the domain, and by exceeding the limitation of pre-defined plan by defining the request by an initial and goal state .

We believe that this is the most appropriate view of automated composition of web services, because the complementation of our models will detect the fault occurences of Web Services by introducing the diagnosis of web service, diagnosis will monitor the execution of the plan, and give the current state of the world. In the case of faults occurence, and according to the state of the world the system will repair the fault by re-planning again.

In our futur works, we will define new metrics for the notion of reliable plan. The degree of reliability of a plan depend of the number of recoverable faults that can occur during the execution of the plan. By calculating this metric we can distinguish the plan the most reliable plan among several plans that can answer the same request .

Finally, we want to enhance our model, by associating the notion of resource reservation to the Web Services and searching for the most appropriate solution to reserving resources for all services constituting a plan.

References

Andrews, T.; Curbera, F. D. H. G. J. K. J. L. F. L. K. R. D. S. D. T. S. I., and Weeravarana, S. 2003. Business process execution language for web services.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. Mbp: a model based planner.

Gottscalk, K., and Team., I. 2000. Web services architecture overview: the next stage of evolution for e-business.

Lazovik, A.; Aiello, M.; and Papazoglou, M. 2003. Planning and monitoring the execution of web service requests.

Malik ghallab, Dana Nau, P. T. Jun 2005. *Automated Planning , theory and practice*.

McIlraith, S., and Son, T. 2002. Adapting golog for composition of semantic web services.

M.Ghallab, A.Howe, C. D. A. M. D. D. 1998. Pddl - the planning domain definition language.

Peer, J. March 22, 2005. Web services composition as ai planning.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. 479–486.

Pistore, M.; Bertoli, P.; Barbon, F.; Shaparau, D.; and Traverso, P. 2004. Planning and monitoring web service composition.

Roberto Chinnici, Martin Gudgin, J.-J. M. J. S. S. W. 2004. Web services description language (wsdl).

Yan, Y., L. Y. L.-H. 2006. Composing business processes with partial observable problem space in web services environment. *Proceedings of the 2006 IEEE International Conference on Web Services. Chicago, septembre 2006.*