Fault Tolerant Web Services Composition as Planning

Dongning Rao[1] Zhihua Jiang[1,2] Yunfei Jiang[1]

[1] Software Research Institute, School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, P.R. China

[2] Department of Computer Science, Jinan University, Guangzhou 510632, P.R. China

Abstract

Fault existence is unavoidable in Web services (WS) providing and calling, so fault tolerant WS has drawn attention of researchers. But while AI planning is known as a promising techniques for WS composition (WSC) problem, few study focus on fault tolerant WSC. So in this paper we first use fault tolerant planning (FTP) approaches to address fault tolerant WSC. We begin by translating WSC into planning problem and than further translate it into FTP, finally we use FTP planners on WSC. We demonstrate our methods through examples in our work, and we believe this work outlines an exciting direction.

Keywords: Web services composition, Fault tolerant, AI planning

1. Introduction

WS are a family of distributed software components that can be exposed and invoked over the internet. This concept was put forward by major IT companies like Microsoft, IBM and Sun as a Web-compatible solution for distributed computing, with the particularly attractive property of being an open, fully standardized and vendor neutral approach. Commonly, the Web Service Description Language (WSDL) [1] is used to describe the syntactical interface of a WS As WS are provided on Web, through socket connections, as black boxes, there are many possibilities of fault, from socket errors, to route issues, and most often the errors or exceptions occurred during services providing. In fact, industry world knows this very well, Microsoft even provides a whole section in MSDN as "Handling and Throwing Exceptions in XML Web Services" [2]. Academic world also aware of the fault existence and a lot of methods have been put forward, e.g. [3]-[6].

The task of automated WSC is to automatically sequence together WS into a composition that achieves some user-defined objectives, and it has received much interest to support business-to-business or enterprise application integration. However, dynamic composition of services is a hard problem and it is not entirely clear which techniques serve WSC best. Academic society draws their attention on WSC as AI planning, where the Planning Domain Definition Language (PDDL) [7] was developed to serve as a standard domain (and problem) specification language. Recently there are a lot of works applying AI planning techniques to WSC, e.g., [8]-[10]. Early works in this research line looked WSC as complex planning action composition as in [9], and recent works as [10] start to consider more real world conditions. But until now no one has considered the failure existence nature of WSC.

The basic insight is there will always be fault or exceptions during calling WS or returned by WS in this flat world. As WSC as planning has been studied for several years and has been proved very successful, our approach is rooted in planning paradigm too. Further more, we believe all the planners have strengths and limitations, to take fault into account one have to use fault tolerant planner. With respect to these facts, we bring fault tolerant planning (FTP) into WSC research, the approach sketched in this paper can work under fault existing environment. To use FTP planners, we have to translate our problem into a FTP problem which should be in NADL+ [11] format. As far as we know, the framework in this paper has never been proposed before, although it builds heavily on previous works.

We have introduced WS and WSC problem, so the remainder of this paper is organized as follows. For a better understanding for WSC as planning and FTP, we focus on explaining them in Section 2. As a major contribution of this work, the whole framework will be introduced in Section 3, and detailed description of its components will be presented in the next three sections. Before summarize, a run-through example will be presented in Section 7. Finally, in Section 8, we summarize the paper and identify some future research directions. We believe there is a whole avenue in front of us in this new frontier research line in WSC.

2. Background and related work

Before move on, we first extend our term and introduce background and related work in WSC as planning research field and FTP research field.

2.1. AI planning

Planning is a complex problem which has been investigated extensively by AI research. [12] characterizes the problem of planning as follows : "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". In general, a planning problem has the following components:

- Descriptions of the possible actions which may be executed (a domain theory) in some formal language;
- Description of the initial state of the world;
- Description of the desired goal;

The formalisms of these components are largely rooted in the STRIPS [13] or the ADL language [14]. Over the time, many AI planning systems have been developed, supporting different levels of expressivity. In many cases these representations are in a middle ground between ADL and STRIPS. To address this problem, the PDDL was developed. Successor versions of the original PDDL version are PDDL 2.1 [15], PDDL 2.2 [16], and PDDL 3.0 [17]. Several other extensions have been proposed, for instance NADL+[11], which is suitable for planning that both explicitly represents uncontrollable environment actions and failure effects of actions; NPDDL[18] which extends PDDL to express nondeterminism, limited sensing and iterative conditional plans; PPDDL [19] which extends PDDL to express probabilities.

The classical view of a plan as a solution to a planning problem is a sequence of operator instances: given a description of an initial state, a goal state, and a set of actions, the planning task is to generate a sequence of actions that, when performed starting in the initial state, will terminate in a goal state. Typically, actions are primitive and are described in terms of their precondition, and (conditional) effects. As classical planning has too many assumptions that make this model far away from real world, nondeterministic planning (NDP) has been devoted to increasing interests. In NDP, actions may have different effects, which effects will become true can only be found out during execution.

2.2. WSC as planning

Currently, WSC is addressed by two orthogonal efforts: the business world developed the Business Process Language for Web Services (BPEL4WS) [21] and the interaction protocols are manually written; academic society draws their attention on WSC as AI planning e.g. [9], [20], [8], [22]-[24], current advance and some open problems are discussed in [25]. Recently, several papers, e.g. [20], have investigated the potentials and boundaries of applying AI planning techniques to WSC. Unfortunately, the planning problem corresponding to the automated WSC is far from trivial, since it poses strong requirements on the kind of planning techniques that can be used. Specifically, it can be hardly addressed by "classical planning" techniques.

[24] says: "By describing a Web service as a process in terms of inputs, outputs, preconditions and effect, using the metaphor of an action, composition can be viewed as a planning problem.", and the same idea is the basement of most papers in this research line, e.g. [9], [20], [22], [23]. Ref. [8] further reveals that web services have unpredicted nature inherit from the internet, so it must be modeled with nondeterministic behaviors, and planning algorithms must work with uncertain effects. In this paper we will follow these works and model Web Services as actions with nondeterministic effects. In the mean time, as far as we know, unlike in WS research line, up to date no one has considered fault existence in WSC. So we further extend these pioneers' works and try to address fault tolerant WSC as FTP.

2.3. FTP

To date, fault tolerant is still very hard for traditional NDP techniques, but we found it can be presented by some recent works as FTP. In [26], they take a first step in this direction by introducing a new class of fault tolerant non-deterministic plans (FTNDP). If there is no fault and the effects of any action are deterministic, than it will never be FTP, so in this paper we use FTP and FTNDP exchangeable. FTP is motivated by two observations:

- Non-determinism in real-world domains is often caused by infrequent errors that make otherwise deterministic actions fail.
- Normally, no actions are guaranteed to succeed.

Due to these observations, FTP proposes a new uncertainty model of action effects in SNDP that distinguishes between primary and secondary effects of actions. The primary effect models the usual deterministic behavior of the action, while the secondary effect models error effects. This definition of fault tolerance is closely connected to fault tolerance concepts in control theory and engineering.

FTP was first defined in [26], as follow:

Definition 1 *(Fault Tolerant Planning Domain) A fault tolerant planning domain is a tuple D = <S;A;T;T' >, where:*

S is the set of states.

A is the set of actions.

T: S×A→ 2S is a deterministic transition relation of primary effects; it associates to each current state s ∈ S and to each action a ∈A the set T(s,a)⊆S of next states.

T': S×A→ 2S is a non-deterministic transition relation of secondary effects.

In [27], they defined plan problem in NDP.

Definition 2 *(plan problem) A plan problem for planning domain D is a tipple <D; I; G>, where:*

D is the planning domain, D= <S;A;T>.

I ∈S is the initial (belief) state.

G ∈S is the goal (belief) state.

We further formalized it in FTP domain:

Definition 3 *(FTP problem) A FTP problem for fault tolerant planning domain D is a tipple <D; I; G>, where:*

D is the fault tolerant planning domain, D= <S;A;T;T'>.

I ∈S is the initial (belief) state.

G ∈S is the goal (belief) state.

Of course if any action fails, the plan will never going to make it, so we set up a upper bound for it, and this definition is the N-FTP problem in [26].

Definition 4 *(n-FTP problem) A n-FTP problem for fault tolerant planning domain D is a tipple <D; I; G;n>, where:*

D is the fault tolerant planning domain, D= <S;A;T;T'>.

I ∈S is the initial (belief) state.

G ∈S is the goal (belief) state.

n is the upper bound on the number of faults the plan must be able to recover from.

[26] also bring algorithms for FTP, but as these are not our contribution we refer readers to their work instead of introduce them, any way that work is far more sophisticated than a short section introduction. Until today, no one uses FTP on WSC or WS research line, we are the first attempt in this direction.

3. Architecture

FTP planners require NADL+ as input, but it is ridiculous to ask WS owner to write a NADL+ description for WS, not even in PDDL format. Most often, WS come along only with its WSDL description. (In fact, you can just append "?wsdl" to the WS's webpage link such as "*.asmx" to view the WSDL description.) So our framework first translates WS description in WSDL format into PDDL format, and then translate PDDL into NADL+, finally we use a planner to solve the problem. Fig.1 shows the flowchart of our framework, and Fig.2 is the system architecture of our system.

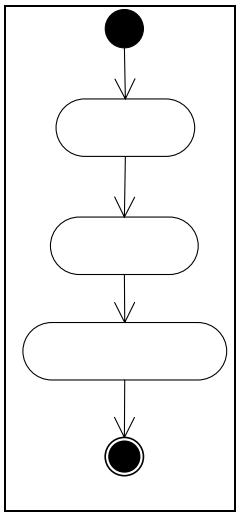

Fig. 1: Framework Flowchart of WSC as FTP.

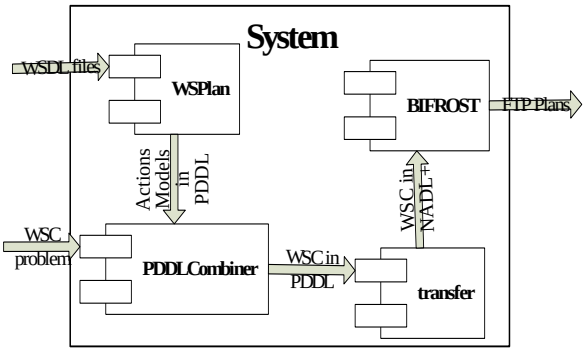

Fig. 2: Framework Architecture of WSC as FTP.

The first component is WSPlan, which is borrowed from [23]. WSPlan extracts/translates action models and system descriptions in PDDL format from WSDL files associated with a WS. we will further introduce WSPlan in Section 4. Then we provide a PDDLCombiner to combine the action models with problem manually provided in PDDL format, it provides interface for human intervene. For the concise of this paper, we choose to ignore the detail implementation for this naïve tool. The third part of our system is a tool presented along with [26], called "transfer", it covers the gap between PDDL and NADL+. The "transfer" becomes one of our basements, and it will be present in Section 5. Finally, we try to find plans with a popular FTP planner, BIFROST, and the planning step will be addressed in Section 6.

4. From WSDL to PDDL

In this section we will introduce the major work of [23], WSPlan. Nowadays, industry world use WSDL to create a Web service contract. WSDL documents do the following:

- Describes what functionality a Web service offers, how it communicates. It describes the abstract interface of a Web service, specifies which operations the service supports, and it defines the format of the messages that must be exchanged to perform the operation.
- Maps an abstract interface to a concrete set of protocols. This mapping is called a "binding", which specifies the technical details of how to communicate with a service.
- Describes a specific Web service implementation. A Web service implementation can support one or more portTypes, each with one or more bindings.

Example 1: A WSDL Definition for a WS

WSDL Definitions

```
(00) <definitions
(01)    xmlns="http://schemas.xmlsoap.org/wsdl/"
…..…
(15) <message name="PurchaseOrder">
(16)          <part          name="Message"
element="msp:PurchaseOrder"/>
(17) </message>
…..…
(33) <portType name="OrderPortType">
(34)  <operation name="ProcessOrder">
(35)   <input
(36)
wsa:Action="http://ex.mspress.microsoft.com/PO"
(37)      message="msp:PurchaseOrder"/>
(38)  <output message="msp: PurchaseOrder "/>
(39)  </operation>
…..…
(50) </portType>
…..…
(70) </definitions>
```

Fig. 3: A WSDL Definition, from MSDN.

A WSDL document containing all three parts describes everything that you need to call a specific WS implementation, see Example 1 (Fig. 3). By [23], this can be seen as operation description.

On the other hemisphere, AI planning researchers provide PDDL as their domain description language. PDDL1.2 [7] supports the following syntactic features:

- Basic STRIPS-style actions
- Conditional effects
- Universal quantification over dynamic universes
- Domain axioms over stratified theories
- Specification of safety constraints
- Specification of hierarchical actions composed of subactions and subgoals
- Management of multiple problems in multiple domains using differing subsets of language features to support sharing of domains across different planners that handle varying levels of expressiveness

Example 2: A PDDL Definition for an Action

```
(:action Lift
:parameters (?x - hoist ?y - crate ?z - surface ?p -
place)
:precondition (and (at ?x ?p) (available ?x) (at ?y ?p)
(on ?y ?z) (clear ?y))
:effect (and (not (at ?y ?p)) (lifting ?x ?y) (not
(clear ?y)) (not (available ?x))
        (clear ?z) (not (on ?y ?z))))
```

Fig. 4: A PDDL Action Definition, for International Planning Competition Benchmark Depot Domain.

An example for PDDL 1.2 is illustrated in Fig.4. Later PDDL 2.1 [15] added the ability of expressing temporal and numeric properties of planning domains, PDDL 2.2 [16] introduced derived predicates and timed initial literals , PDDL 3.0 [17] involved trajectory and preference. As the first step towards this way, we only consider PDDL 1.2, so does [23].

WSDL and PDDL are coming from different planet, they have their own strength and weakness, to transform form WSDL into PDDL is a great work more than this short section can covered. Here we just briefly sketch [23]'s idea, and refer readers to [23] for more detail.

Fig. 5 shows relevant parts of the conceptual models of WSDL and PDDL. Bridging by an annotation format (in the middle), WSPlan builds connections between these two worlds, and finally transfers WS WSDL description into action models in PDDL. Each box in the illustration represents an element of the WSDL 1.2 model, annotation model or the PDDL model, respectively. The solid lines represent concept associations and the dashed lines represent the semantic associations captured by our service annotation concept. We are not going to further introduce this tool, but any way, following their steps, we have PDDL format descriptions for WS.

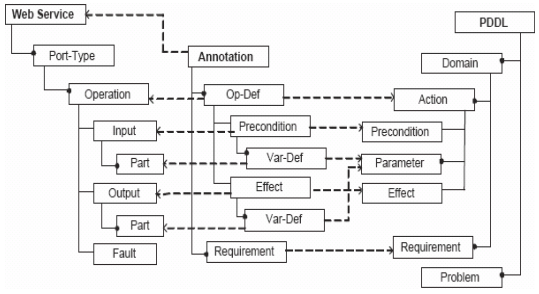

Fig. 5: WSDL to PDDL, from [23].

This translation result for Example 1(Fig. 3) is presented in Example 3 (Fig. 6).

Example 3: PDDL Description for Example 1

(define (domain ORDERS)
(:requirements ::strips :typing :equality :conditional-effects)
……
(:action ProcessOrder
:parameters (?message0 ?)
:precondition (and ())
:effect
(and
(when (not (Received ?message0))
(Received ?message0))
(when (and (Received ?message0)(not (Received ?message1))
(Received ?message1))
(when (and (Received ?message1)(not (Received ?message2))
and (Received ?message2)(ORDER))
……

Fig. 6: PDDL Representation for Fig.3.

5. From PDDL to NADL+

Before calling FTP planner, the system and problem should be transferred into NADL+ format, so we introduce "transfer" and how to transfer PDDL to NADL+ in this section. NADL+ is one of the successors of ADL, it was developed as a part of the UMOP project [29]. However, despite providing a very general framework for modeling non-deterministic planning problems, it does not allow additional information about transition costs, heuristic estimates, and failure effects of actions. Hence later NADL+ was developed and adds these features to the language. According to [29], there are three main differences between the two languages:

- NADL+ has three new optional action description components dg, dh, and err. It uses the entry heu to define the value of the heuristic estimate in the initial state and the goal states,
- Action descriptions may consist of descriptions of several transition groups,
- NADL+ assumes that the system and environment are described by as set of actions.

There are also three action components:

- dg: int associates a transition cost or weight with the action.
- dh: int describes the change of a heuristic estimate associated with each transition represented by the transition group. The change is always given in forward direction even if the heuristic guides a backward search.
- err: formula defines a set of next states reached by the action given that its execution fails.

An NADL+ problem description consists of:

- A set of state variables;
- A set of system and environment actions;
- An initial and goal condition.

As traditionally in planning community, the set of state variable assignments defines the state space of the domain; the set of system actions must be non-empty while the set of environment actions may be empty if no active environment exists; each step, exactly a single system and environment action is performed.

An action has three main parts:

- A set of modified state variables;
- A precondition formula;
- An effect formula.

As traditionally in planning community, the set of modified variables are the state variables which may have their value changed by the action; in order for an action to be applicable, the precondition formula must be satisfied in the current state; the effect of the action is defined by the effect formula. The value of state variables not modified by a joint action is unchanged. The initial and goal condition are formulas that must be satisfied in the initial state and the goal states, respectively. We refer readers to [29] for more detail syntax, here we just give an example(Fig. 7).

Example 4: A NADL+ Description of Action

pick_upBlock1
con: posblock1
pre: posblock1 = 1 /\ posblock2 > 0 /\ posblock2 <> 2 /\ posblock3 > 0 /\ posblock3 <> 2 /\ posblock4 > 0 /\ posblock4 <> 2
eff: posblock1' = 0

Fig. 7: A NADL+ Action Definition, for International Planning Competition Benchmark BlocksWorld Domain.

Along with NADL+, [29] provide a tool called "transfer" to translate between PDDL and NADL+. If the reader is interested in "transfer", further details are

on the web for this open source project. After using "transfer", we get results as in Example 5 (Fig. 8).

Example 5: NADL+ Representation for Example 1

```
VARIABLES
  nat(10) message;
  bool order;
SYSTEM
  agt: sys
   processorder
    mod: message
    pre: message = 0
eff: message = message + 1
err: false
    mod: message
    pre: message = 1
eff: message = message + 1
err: false
    mod: message, order
    pre: message = 2
eff: message = message + 1/\ order=true
err: false
INITIALLY
   message = 0 /\ ~ order
GOAL
   message = 3 /\ order
```

Fig. 8: NADL+ Representation of Example 1

6. Planning

The last step is planning, which we are going to state in this section. BIFROST is the Bdd-based InFoRmed planning and cOntroller Synthesis Tool (BIFROST). BIFROST version 0.7 is a software package for BDD-based deterministic and non-deterministic planning and heuristic search. The program is written in C++/STL for the GNU GCC compiler running on a Redhat Linux 7.1 PC. BIFROST uses the BuDDy 2.0 BDD-package [28].The input to BIFROST is a planning problem written either in the STRIPS part of PDDL or NADL+. Simply put, BIFROST can use NADL+ as input and generate FTP plan.

To avoid tedious jobs of introducing this open source tool, we use examples instead of detail operation manuals here.

Example 6: Using BIFROST

First prepare the following command line:

```
bifrost -i NADL -d D4V4M15.nadl -g MinHamming -l
500 -u 200 -n 8000000 -c 700000 -x 1.0 -y 1.0 -t 5000
-e ghSetAstar.exp -a ghSetAstar
```

Then run it under Linux enviorment

Fig. 9: Calling Example of BIFROST, from [29].

Then we can get our planning result as :

Example 6: Result from BIFROST (1-FTP)

```
ProcessOrder0
ProcessOrder1
ProcessOrder2
ProcessOrder2
```

Fig. 10: Result of Planning

7. A work-through example

In the following example we will outline the different aspects of our framework and the transform algorithm. The following sample was adopted from Microsoft website, and most of its steps have been introduced in prior examples.

Sample:

In an E-Shop, there are three phases of a shopping, first order, then review, and finally confirm. Whatsoever, the website will confirm a purchase after received three PurchaseOrder messages, if there are less than three messages, it would drop them after a time limitation.

Step 1: Get the WSDL for Web service.

See Example 1, Fig. 3.

Step 2 and Step 3: Use WSPlan method to transform it into PDDL format and then manually modify it. WSC problem specification should be provided in PDDL here, too.

During step 2, there has some manual work. First of all, the domain name has to be assigned, along with every action name, because without any semantic reasoning ability WSPlan uses meaningless name to avoid conflict. Second, every domain description in PDDL must be given a set of predicates, but "This is needed to connect the predicates and constants to semantic web ontologies. However, WSPlan does not process ontological information yet." [23], so this work has to be done by hand. Last but not least, every service must get something done, for example here ProcessOrder suppose to relate a customer and a book, and this underlining semantic issue can never be covered by syntax transfer alone.

Any way, after translation, we have a PDDL file like Example 3, Fig. 6. Of course, if we have experts to rewrite it, we can have err effects other than "false", which means nothing happened to the world.

Step 4: Use "translator" provided by BIFROST to translate this into NADL+.

It is our duty to notice that NADL+ has instantiated all actions, which means it is very big, and hence it required initial states and goal to be written down before translation. Unfortunately, this translated script can not used directly as fault tolerant planning domain description. So we use flex to make a small tool for it, and translate it in to FTP domain. The result is like Example 4, Fig. 8.

Step 5: Use BIFROST to find a solution.

See Example 6, Fig. 9. and Example 7, Fig. 10.

8. Conclusions and future work

In this paper, we focus on WSC with fault tolerance, which is more suitable for real world applications, and the framework we presented can be seen as the core contribution of this paper. As a difference with previous work done in this direction, we first translate WSC problem into NADL+ format, which means it can be solved by FTP planners like BIFROST. This process can in principle be reused for other non-deterministic domain description and NDP planner with no modifications whatsoever. Here are the main points we tried to make:

- WS is born with non-deterministic nature, so WSC should be handled in NDP scope.
- Fault existence is unavoidable in web service providing and calling, it should be put on researchers table at the beginning.
- Every planner has its strength and weakness, if we want to make fault tolerant plan, we have to use fault tolerant planner.
- PDDL is not suitable for web service description in WSC.

Based on the above insights, and stand on giants' shoulders, we investigate the potential of merging the best characteristics of AI planning, especially NDP and FTP, and WS in real world. To do so, we first translate WSC into planning problem. This is done by a semi-auto method, which builds heavily on [23]'s work to translate WSDL into PDDL. And then we further translate this PDDL file into NADL+ format, by a tool called "transfer" provided by [29]. Finally we use a FTP planner called BIFROST to address WSC. We also demonstrate our methods through examples in our work. We believe this framework can be benefited from any advantages in either NDP or WS research line.

One of the currently prominent service composition planners is SHOP2 in [8], one of SHOP2's shortcomings is not allowing fault existence. Comparing to WSC with SHOP2, our method can provide fault tolerant ability. Another great work which is also one of this paper's stepping-stones is [23]. But [23] works with deterministic planning and we believe WS has a non-deterministic nature. So we use non-deterministic domain description, instead of PDDL for it.

Combining two research lines' best characters is very promising, and the results are often exciting. But, this is by no means the end of the story. Here we note some current limitations of our approach. First, the semantic differences can be caused by many factors, and to cover it is a crucial step toward fully automatic translation. We prepare to set about combining semantic match up later. Second, real world applications have complex natures like synchronized communication, and most of the WS today is stateless. We plan to modify BIFROST to a synchronized one to cover this problem. Third, planners have suffered a lot from scale issues for a long time, but if it is used in real world, the search space will grow exponentially. Considering that the formulation of planning as heuristic search with heuristics derived from problem representations has turned out to be a fruitful approach for classical planning, we are testing some heuristic strategies on BIFROST.

Acknowledgement

This work is partially supported by National Nature Science Foundation of China.

References

[1] W3C, Web Service Description Language (WSDL) version 1.2 (2002)

[2] http://msdn2.microsoft.com/en-us/library/ds492xtk(VS.71).aspx

[3] N. Aghdaie and Y. Tamir, Implementation and Evaluation of Transparent Fault-Tolerant Web Service with Kernel-Level Support *In Proc. of the IEEE International Conference on Computer Communications and Networks*, pp. 63-68, October 2002.

[4] D. Liang, C. L. Fang, C. Chen and F. Lin, Fault tolerant Web service. *Software Engineering Conference, 2003*, 10:310 – 319, 2003

[5] Y. Lee, J. Oh, and S.Han. Enriching Quality and Fault-Tolerance of Web Services System. *International Journal of Web Services Practices*, 1:153-157, 2005

[6] L. Ardissono, R. Furnari, A. Goy, G. Petrone and M. Segnan, Fault Tolerant Web Service Orchestration by Means of Diagnosis, *In Proc. of EWSA 2006* pp. 2-16, 2006

[7] M. Ghallab, A. Howe, C. Knoblock, , D. McDermott, A. Ram, M. Veloso, D. Weld and D. Wilkins, PDDL the planning domain definition language, 1998.

[8] D. Wu, E. Sirin, J. Hendler, D. Nau and B. Parsia, Automatic Web Services Composition Using SHOP2, *In Proc. of the Twelfth International World Wide Web Conference (WWW2003)*, May 2003.

[9] S. McIlraith and R. Fadel, Planning with Complex Actions, *In Proc. of International Workshop on Non-Monotonic Reasoning*, 2002

[10] M. Pistore, P. Traverso and P. Bertoli, Automated Composition of Web Services by

Planning in Asynchronous Domains, *In Proc. of ICAPS 2005*, pp.2-11, 2005.

[11] R. M. Jensen and M. M. Veloso, OBDD-based deterministic planning using the UMOP planning framework, *In Proc. of the AIPS-00 Workshop on Model-Theoretic Approaches to Planning*, pp.26–31, 2000.

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

[13] R. E. Fikes and N. J. Nilsson, STRIPS: A new approach to theorem proving in problem solving, *Artificial Intelligence*, 1971.

[14] E. Pednault, ADL and the state-transition model of action, *Journal of Logic and Computation*, 1994.

[15] M. Fox and D. Long, PDDL 2.1: An extension to pddl for expressing temporal planning domains, *Journal of Artificial Intelligence Research*, 20, 2003.

[16] S. Edelkamp and J. Hoffmann, PDDL2.2: the Language for the classical part of the 4th International Planning Competition, Albert Ludwigs Universitat, Institut fur Informatik, Freiburg, Germany, *technical Report: 195*, 2003.

[17] A. Gerevini and D. Long, Plan constraints and preferences in PDDL3: The language of the fifth international planning competition, *Technical report*, University of Brescia, Italy, 2005.

[18] P. Bertoli, A. Cimatti, U. Dal Lago and M. Pistore. Extending PDDL to nondeterminism, limited sensing and iterative conditional plans, *In Proc. of ICAPS Workshop on PDDL, Informal Proceedings*, pp.15-24, 2003.

[19] H. L. S. Younes and M. L. Littman, PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects, *Tech. rep. CMU-CS-04-167*, Carnegie Mellon University, Pittsburgh, PA, 2004.

[20] D. McDermott, Estimated-regression planning for interactions with web services, *In Proc. of the AI Planning Systems Conference 2002*, *AAAI*, 2002.

[21] IBM, Microsoft and BEA, Web Services Business Process Execution Language Version 1.0 2002.

[22] E. Martínez and Y. Lespérance, Web Service Composition as a Planning Task: Experiments using Knowledge-Based Planning, *In Proc. of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services*, pp.62-69, 2004

[23] J. Peer, A PDDL based Tool for Automatic Web Service Composition, *In Proc. of the Second Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2004) at the 20th International Conference on Logic Programming*, LNCS 3208, 2004.

[24] M. Vukovic and P. Robinson, Adaptive Planning Based Web Service Composition for Context Awareness, *In Advances in Pervasive Computing*, 176:247–252, 2004.

[25] B. Srivastava and J. Koehler, Web Service Composition - Current Solutions and Open Problems, *In Proc. of ICAPS 2003 Workshop on Planning for Web Services*, 2003.

[26] R.M. Jensen, M. Veloso and R.E. Bryant, Fault Tolerant Planning: Toward Probabilistic Uncertainty Models in Symbolic Non-Deterministic Planning *In Proc. of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pp.335-344, 2004.

[27] A. Cimatti, M. Pistore, M. Roveri and P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, *Artificial Intelligence*, 147:35--84, 2003.

[28] J. Lind-Nielsen, BuDDy - A Binary Decision Diagram Package, *Technical Report IT-TR: 1999-028*, Institute of Information Technology, Technical University of Denmark, 1999. (http://cs.it.dtu.dk/buddy)

[29] R. M. Jensen, Efficient BDD-Based Planning for Non-Deterministic, Fault Tolerant, and Adversarial Domains. *Ph.D. thesis, Carnegie Mellon University, CMUCS-03-139*. 2003.