# Web Service Composition as AI Planning – a Survey[*]

Joachim Peer

March 22, 2005

**Abstract**

This article gives an overview of AI (Artificial Intelligence) planning techniques and discusses their application to the Web service composition problem.

[*]Second, revised version, March 2005.

# Contents

**1 Introduction and Motivation** **3**

**2 Scenarios** **5**

**3 Preliminaries** **6**

3.1 Formalizing the planning domain . . . . . . . . . . . . . . . . 6

3.2 Formalizing the Initial World . . . . . . . . . . . . . . . . . 9

3.3 Formalizing Goals . . . . . . . . . . . . . . . . . . . . . . . . 11

3.4 Representing Plans . . . . . . . . . . . . . . . . . . . . . . . 13

**4 Basic Planning Paradigms** **14**

4.1 State-Space based Planning . . . . . . . . . . . . . . . . . . . 14

4.2 Graph Based Planning . . . . . . . . . . . . . . . . . . . . . . 18

4.3 Partial Order Refinement Planning . . . . . . . . . . . . . . . 22

4.4 Planning as Satisfiability . . . . . . . . . . . . . . . . . . . . 26

4.4.1 Planning as Propositional Satisfiability . . . . . . . . . 27

4.4.2 Planning as Description Logic Satisfiability . . . . . . . 28

4.4.3 Planning as Petri-Net Reachability . . . . . . . . . . . 29

4.5 Planning as Logic Programming . . . . . . . . . . . . . . . . 30

**5 Planning with Control Knowledge** **31**

5.1 Hierarchical Task Network Planning . . . . . . . . . . . . . . 31

5.2 High-level Program Execution . . . . . . . . . . . . . . . . . 33

5.3 Planning As Model Checking . . . . . . . . . . . . . . . . . . 36

5.4 Temporal Planning . . . . . . . . . . . . . . . . . . . . . . . 39

**6 Discussion and Outlook** **41**

# 1 Introduction and Motivation

Web services are distributed software components that can be exposed and invoked over the internet using standard protocols. 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.

Web services communicate with their clients and with other Web services by sending XML based messages over the internet. The signatures of the operations a Web service offers and the message formats it supports form its *interface*. Commonly, interface description languages (IDLs) such as the Web Service Description Language WSDL (W3C, 2002) are used to describe the service interface.

WSDL allows for decoupling abstract descriptions of service types (called *Port Types*) from concrete implementations of the services. Therefore, a single Port Type description can be used for multiple services of similar type. This allows for the definition of *standardized* service interfaces, and participants with a common interest can jointly reach agreements on the semantics of those descriptions. Based on such agreements, client applications may be crafted to use the Web services, and complex processes involving several services may be composed, for instance using the BPEL4WS (IBM et al., 2002) process description language.

A problem of this approach becomes immanent when services diverge from the initial agreements. For instance, when a service changes its implementation (e.g. to refine its service offerings) its semantics and probably its syntactic interface will change. Since there is no formal machine interpretable connection defined between the semantics and the syntactic interface, human intervention is needed to decide whether the service is still compatible with the agreed semantics or not.

A way of addressing this limitation is writing down a sufficiently large part of the semantics of a service in a *formal machine interpretable* fashion, quite analogous to the syntactic interface. This reduces the dependency on external semantic agreements that are often difficult to reach, must be reached for each new service type, and must be re-evaluated after each modification of a Web service. Instead, the semantic descriptions provide software agents with a way to autonomically reason about the service's semantics, i.e. the preconditions and consequences of its operations.

In an environment of semantically annotated services, users who need to achieve certain goals could be assisted by software agents which automatically identify and, if necessary, dynamically compose services in order to accomplish the user's goals, which may be either explicitly stated or derived from the situation the user is in.

However, dynamic composition of services is a hard problem and it is not entirely clear which techniques serve the problem best. One family of techniques that has been proposed for this task is AI (Artifical Intelligence) planning. Planning is a complex problem which has been investigated extensively by AI research. (Russel and Norvig, 1995) characterize 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".

Recently, several papers, e.g. (McDermott, 2002; Srivastava and Koehler, 2003; Carman et al., 2003; Sirin and Parsia, 2004), have investigated the potentials and boundaries of applying *AI planning techniques* to derive web service processes that achieve the desired goals. In this report we aim to extend this research by providing a survey of the most important planning techniques and by discussing their suitability for dynamic Web service composition.

The remainder of the article is organized as follows: in Sect. 2 we list the scenarios we gather our requirements from. In Sect. 3 we discuss the relevant conceptual frameworks of planning. We then proceed to the discussion of basic planning paradigms in Sect. 4 and knowledge oriented planning paradigms in Sect. 5. We then contrast a collection of representative planning engines against our identified core requirements and discuss the results and possible directions for future work.

## 2 Scenarios

To assess the importance of the various potential requirements we are confronted with, we consider the following collection of Web service domains:

- The *Web shopping* domain (Peer, 2004b): A collection of services that offer capabilities to browse catalogs and purchase goods. Possible goals are to find and purchase one or more products, possibly at best price.
- The *document handling* domain (Peer, 2004b), which is similar to the Softbots domain in (Golden, 1997): The services offer functions to manipulate files, for instance to convert, compress or encrypt them. Possible goals are series of document transformations, e.g. to convert and package a collection of documents.
- The *mail replication* domain (Vukovic and Robinson, 2004): This domain combines electronic mail-related services, i.e. SOAP interfaces to SMTP and POP servers, with services in the document handling domain (cf. above). Typical goals in this domain are the sending and receiving of messages, and the context depended adaption of the behavior of the mail system, e.g. context- and user-dependent display, involving text translation and image conversion services.
- The *traveling* domain (McIlraith and Son, 2002): The services in this domain offer the capability to query and book air tickets and accom-

modation for travelers. A typical problem of this domain is to plan for a conference attendance, which often involves additional user-defined constraints to be satisfied (e.g. the date of the conference, preferences for certain hotels or airlines).

# 3 Preliminaries

In general, a planning problem has the following components:

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

In the following sections, we present the most important approaches to define the components of a planning problem and we will contrast them with the requirements of Web service composition problems.

## 3.1 Formalizing the planning domain

The aim of domain formalization is to provide a *domain theory*, i.e. a formal account of the semantics of the operations that are available or relevant to the agent. These operations can represent physical operations (e.g. defined by a robot's physical environment) but can also represent more abstract actions (e.g. withdrawing money from a bank account).

A domain theory must formally define the *causal laws* of the operations, i.e. it must allow to axiomatize relevant aspects such as the preconditions of operations and their effects to the world. Usually, domain theories follow some state-transition model, i.e. they introduce a notion of *state* (or situation), which is a snapshot that describes the world at a certain point in time and they relate actions to transitions between such states. Most approaches define a state extensionally as a set of ground atomic formulas (atoms), where atoms that may change their value over time are called *fluents* and those that do not change are called *state invariants*.

Regarding the epistemological principles domain theories are based on, we can distinguish two variants: domain theories based on classical logics and extra-logical domain theories.

Among the logical approaches is the situation calculus, which was introduced by (McCarthy, 1963) and later refined by (Levesque et al., 1998; Pirri and Reiter, 1999), who define the situation calculus as a second-order framework designed for representing dynamically changing worlds in classical first-order language. The situation calculus represents the world and its change as sequence of situations, where each situation is a term that represents a state and is obtained by executing an action[1].

Another approach for encoding operations into first order predicate logics is the *event calculus* (Kowalski and Sergot, 1989). In the event calculus, events initiate periods during which certain properties hold. A property is initiated by an event and continues to hold until some event occurs that terminates it. The events, their effects and durations are expressed in Horn logic.

Yet another approach based on logic are action theories based on modal logics, as discussed by (Giacomo and Lenzerini, 1995; Castilho et al., 1999; Giordano et al., 1998). Like the situation calculus, the modal approaches define a system of world states where the actions are modeled as transitions between those states. Modal logic approaches allow for a very natural modeling of actions as state transitions, by conceptualizing them as *accessibility relations* in Kripke structures. As we will see later in Sect. 5.3, Kripke structures are indeed practically used to represent nondeterministic domains.

Despite the advantages of these pure logic based approaches, such as the

[1]A more detailed description of the situation calculus is presented in Sect. 5.2

precise semantics and the ability to prove certain properties of domain theories, the AI planning community largely uses different formalisms to express planning domains. These formalisms are largely rooted in the STRIPS (Fikes and Nilsson, 1971) notation, which was used in the 1970ies to describe planning domains for a robot system called "Shakey". STRIPS allows to define operators directly by specifying a precondition, an ADD-list and a DELETE-list, all represented as conjunctions of atoms. Intuitively, the semantics of such an operator description is that an operation is only applicable if the precondition is satisfied by the current world state (represented as a database), and that after execution of the operation the atoms of the ADD-list will be added to the world state and the atoms of the DELETE-list will be removed. However, the precise logical semantics of STRIPS has been a subject of debate for long time, with different proposals put forward, e.g. (Lifschitz, 1986; Reiter, 2001).

The ADL language (Pednault, 1989; Pednault, 1994) provides support for more expressive operator descriptions and narrows the gap between the semantically ambiguous STRIPS and the declarative situation calculus: ADL allows the definition of context dependent effects, universally quantified effects (for instance needed to model the transportation of goods using a truck), negation and disjunction.

Over the time, many AI planning systems have been developed, supporting different levels of expressivity, in many cases in a middle ground between ADL and STRIPS, sometimes even beyond, e.g. to express temporal reasoning, metrics, task networks, etc. This resulted in a wide range of "ad-hoc" formats, whose semantics have often been ambiguous. To address this problem, the Planning Domain Definition Language (PDDL) (Ghallab et al., 1998) was developed to serve as a standard domain (and problem) specification language, to ease the comparison of the various systems. PDDL allows to define domains of the expressivity of ADL, including metric fluents, and defines rules for standard-compliant extensions. Successor versions of the original PDDL version are PDDL 2.1 (Fox and Long, 2003) which added a notion of time and PDDL 2.2 which adds derived predicates and timed initial literals. Several other extensions have been proposed, for instance (Bertoli et al., 2003) which extends PDDL to express nondeterminism, limited sensing and iterative conditional plans.

## 3.2 Formalizing the Initial World

A planning agent must take the initial world state into account, because it must provide a plan that, when executed in the initial world, will lead to the specified goal.

The conceptualizations discussed in the last section not only define the conceptual models of actions, they also define the conceptual models of the initial world description a planning agent is given. In fact, the initial world is just another world state (or situation) defined by the domain theory.

The central element that constitutes a world state in practically all approaches are the atoms that are known to be true in the initial world state. Classical AI planning approaches assume that the extensional definition of the initial world state provides a *complete* description. This allows to employ the closed world assumption, which means that any fact that is not explicitly listed in the state database is false.

For real world applications, such as in robotics or in our domains of Web service computing, these simplifying assumptions are unrealistic. In fact, we are confronted with the following problems:

- Incomplete information: the extensional definition of the initial world does not specify all knowledge relevant to the planning task. For instance, in an e-commerce application, the agent may not know which online retailer offers which products, but it needs this information to achieve its goal of *buying* a product.

- Wrong information: some of the atoms that are defined as true may be false in reality (and vice versa). This happens when the *invocation and reasonable persistence (IRP)* assumption (McIlraith and Son, 2001) is violated, i.e. when facts are changed after the agent has acquired knowledge about those facts and when the agent wrongly believes its knowledge is accurate.

- Fuzzy information: for each known fluent value there might exist a certain probability that it is not correct (e.g. because of fuzzy sensors). Again, this problem does not appear frequently in our domains.

The conceptual models of planners have been extended over the time to better deal with the difficulties listed above. Since the world view of an agent may divert from the reality of the world, it is useful to explicitly represent the knowledge an agent has. The agent's knowledge can be constituted by the knowledge of atomic facts and also certain axioms and functions. Along these lines, alternatives to the widely used closed world assumption have been investigated, for instance the *Local Closed World Assumption*(LCA), which allows to represent Local Closed World (LCW) knowledge (Golden, 1997). LCW knowledge is usually organized in two databases $\mathcal{M}$ and $\mathcal{L}$, where the database $\mathcal{M}$ contains a collection of all known facts and the database $\mathcal{L}$ contains LCW formulas that describe the contents of $\mathcal{M}$, i.e. they state for which parts of the world the agent's knowledge can be safely assumed to be complete. For instance, when an agent queries the list of all products an online retailer $A$ sells, it may assume to know all products that are available from $A$ (when the IRP assumption holds).

When knowledge of agents is expressed explicitly, the necessity arises to define the influence some of the domain's operators have on the agent's knowledge. In other words, it is useful to distinguish operators (or effects) that change the *world* from operators (or effects) that only affect the agent's *knowledge*. The latter are called sensing operations (or sensing effects). An extension to STRIPS that accounts for incomplete knowledge and sensing actions is UWL (Etzioni et al., 1992). Similarly, SADL (Golden, 1997) adds support for incomplete information and sensing to ADL, and NPDDL (Bertoli et al., 2003) proposes similar extensions to PDDL. An other proposed extension to PDDL is Opt (McDermott, 2002), which adds knowledge effects and *learnable terms* to the PDDL framework.

A formal situation calculus based account of the incomplete knowledge of agents and sensing actions was given in (Moore, 1985), which also introduced the notion of knowledge preconditions, which are conditions the agent's knowledge base must fulfill to successfully apply an operator.

## 3.3 Formalizing Goals

In most classical approaches to AI planning, goals are expressed as properties that need to hold in a desired world state (the *goal state*), usually in the form of conjunctions and disjunctions of literals (positive or negative atoms) and whereby variables are treated with existential quantification.

The planner needs to identify a solution (a plan), which, when executed in the initial world state, will result in a world state that satisfies the goal. For instance a goal *(color Door1 Red)* specifies a condition that says that the fluent *color* of *Door1* must have the value *Red* after plan execution, and a goal *(have-door House1 ?d)* would require the existence of an constant $c$, that when bound to variable $?d$, would make the formula *(have-door House1 ?d)* true.

For automated Web service composition (and many other domains) these goals specifications are not sufficient enough. Requirements listed in the literature are:

- Need for temporal structures: Certain complex goals can not be expressed simply as properties of a final state. For instance, the planning of a round-trip from Vienna to Zurich and back can not be expressed as a condition on the goal state because the goal state would equal the initial state (i.e. the agent being in Vienna[2]). Therefore, certain structures need to be added to split the planning goal into several distinct, consecutive phases. In some cases, a valid plan may even have to include looping and branching, as noted by (Srivastava and Koehler, 2003).

- Strategies for dealing with nondeterminism, i.e. how to behave if the execution of an operation does not achieve the expected or desired result (e.g. by defining BPEL-like *compensation actions*).

- Safety properties: not all *possible* solutions to achieve a goal are *desired* ones. For instance, there may be certain fluents whose values should not or only moderately be changed (e.g. the credit-card balance); these protected fluents are sometimes called *maintenance goals* or *resource constraints*. Further, in many domains there are certain situations an agent may stumble into, which need to be avoided altogether, and constraints on the goal can help to evade them.

- Distinction between *information goals* and *achievement goals*: Many problems require such a distinction because information goals should be achieved exclusively by *sensing actions*. As an example, a goal to find out the current color of an item may only use operations that do *not* actively affect its color; we would not want the agent to use an operation that sets the color to some new value and then reports that newly assigned color (Golden, 1997). Instead, the value should be gathered through a *sensing operation*.

- Preferences of users over possible solutions (e.g. preferring air travel over traveling by train, payment via credit card over e-cash) and other user-provided constraints on the solution (e.g. buying airline ticket

[2]although the problem could be circumvented by formulating the goal as possessing a ticket from Vienna to Zurich and a second ticket from Zurich to Vienna.

only if driving would take longer than 3 hours) (McIlraith and Son, 2001)

Since these difficulties are relevant to many real world planning domains, not only to Web service composition, there exist several approaches to address these problems, which will be discussed in Sect. 5.

## 3.4 Representing Plans

The classical view of a plan as a solution to a planning problem is a *sequence of operator instances*, which, when executed leads to a state that satisfies the given goal. Given the discussion of goals in the last section, especially the problem of nondeterminism, it is not surprising that this classical view of plans is not always sufficient to capture the solutions to complex planning problems.

The required complexity of a plan does not only depend on the domain and goal complexity, it also depends on the execution model foreseen for the plan: if an operation does not yield the desired result, will the agent have the opportunity to re-generate the plan (as in replanning/reactive planning architectures, e.g. (Firby, 1987)), or will the agent have to rely on the predefined plan? In the latter case, a *conditional* plan is required that deals with the nondeterminism and incomplete information by constructing a plan that accounts for the possible contingencies that could arise. At runtime, the agent has to determine the situation it is in and then chose the appropriate plan branch that is prepared for that situation. Planners that adopt that strategy are also called contingency planners.

Beside contingency planners, there exist several other extensions to plans as simple sequences. As we will discuss later in Sect. 4.3, partial order planners allow for plans whose actions are partially ordered, i.e. some of the actions can be executed in parallel rather than sequentially, which often increases the efficiency of the system. Even more feature-rich plans can be created using the planning as model-checking (MC) approach described in Sect. 5.3, where the planner synthesizes plans that may contain loops and branches.

# 4 Basic Planning Paradigms

In the following we will give an overview of the basic planning paradigms and some representative implementations of these concepts.

## 4.1 State-Space based Planning

A state space consists of a finite set of states $S$, a finite set of actions $A$, a state transition function $f$ that describes how actions map one state into another, and a cost function $c(a, s) > 0$ that measures the cost of performing action $a$ in state $s$ (Fikes and Nilsson, 1971). A state space extended with a given initial state $s_0$ and a set $S_G$ of goals is also called a *state model* (Bonet and Geffner, 2001b).

State based planners aim to solve a planning problem by searching for useful operator instantiations that achieve the desired state. Depending on the starting point of the search, we distinguish forward state search (also called progression) and backward state search (also called regression): A progressive state based planner starts with the initial state and searches action instances that bring the planner closer to the goal. A regression planner starts with a state satisfying the goal and searches for action instances that bring the planner closer to the initial state.

In both cases, the goal is to find a sequence of actions that, when applied beginning in the initial state, will lead to the goal state. More formally, a solution of a state model is a sequence of actions $a_0, a_1, ..., a_n$ that generates a state trajectory $s_0, s_1 = f(s_0), ..., s_{n+1} = f(a_n, s_n)$ such that each action $a_i$ is applicable in $s_i$ and $s_{n+1}$ is a goal state, i.e., $a_i \in A(s_i)$ and $s_{n+1} \in G$

(Bonet and Geffner, 2001b).

In principal, any search algorithm can be used to perform state based search, and the discipline of means-end-analysis has a long tradition with roots in the 1950s since GPS (Newell and Simon, 1963). However, the usually vast number of different branches of actions a planner has to chose from calls for methods that reduce the search space or help discriminating fruitful vs. useless branches of the search tree. An early attempt for reducing the search space was the STRIPS algorithm. It uses backward search, i.e. it starts with the goal, searches an action that achieves the goal or one of its subgoals and then goes on to search actions that achieve the precondition of the actions, and so forth. STRIPS enhances this search by only considering the preconditions of the last operator added to the plan and by committing to operators whose preconditions are satisfied by the current state. This reduces the plan space significantly, but it makes STRIPS incomplete, i.e. there is no guarantee that a solution for a problem will be found even if there exists one.

A different way of dealing with the vastness of the plan space is to employ *heuristic functions* which estimate the usefulness of the alternative actions a planner can chose from, thus *guiding* the planner to chose fruitful search paths and ignore branches that will lead to dead ends or solutions of low quality. Truly automatic domain independent planners have no other choice than gathering these heuristics from the domain and the problem descriptions they are confronted with, in contrast to specialized algorithms, e.g. the algorithm solving the 8-puzzle discussed in AI textbooks like (Nilsson, 1980).

A planner adopting such a domain-independent heuristic is UNPOP (McDermott, 1996), which employs a regression-match graph. The construction of this graph starts with the goal, which is matched to the current situation, yielding a set of literals to be achieved. In the next level of the graph, actions are considered that achieve some of those subgoals, which yields another set of subgoals, needed to carry out those actions. Those subgoals are added to the next level of the graph and the process repeats. To enhance the graph traversal, the notion of *estimated effort* is used, i.e. an estimate of how many actions it will take to achieve the main goal, whereby the effort of a goal that is already given in the current situation is 0 and the effort of a goal that can not be achieved by any operator in the domain is $\infty$. When traversing the regression graph, UNPOP chooses branches first whose effort estimations seem favorable, which leads to an improvement in planning speed compared to “blind” unguided searches.

In related work, the forward planner HSP (Heuristic search planner) was presented, which is based on the *additive heuristic* $h_{add}$. This heuristic defines the cost of a set $C$ of atoms as the sum of the cost of the elements of $C$. This assumes that subgoals are independent of each other, which is not always true because some goals can become less (or even more) difficult once other goals are fulfilled. The HSP system uses $h_{add}$ to guide a *hill-climbing search* from the initial state to the goal. At each step, one of the best child nodes (i.e. nodes whose $h_{add}$ value is minimal) is selected for expansion and the same process is repeated until the goal is reached. The costs are calculated as *estimations*, which are extracted from a relaxed planning problem $P'$, where the negative effects of operators are ignored. The estimations are generated by iteratively applying the positive effects of a number of operations whose preconditions are applicable in the current state (negative effects are ignored) and by tracking for each atom that is achieved in that process after how many steps it was achieved (Bonet and Geffner, 1998). Empirical data shows that the idea of using a relaxed problem to harvest heuristic estimation as well as the assumption of goal independence yields preferable results, as documented in the results of the international planning competition IPC-1998 (McDermott, 2000; Bonet and Geffner, 2001b).

In successive work, the planner HSP2 (Bonet and Geffner, 2001a) was developed, which employs the same heuristic function $h_{add}$, but uses *best-first search* (Pearl, 1985) instead of hill-climbing. The best-first search weighs nodes by an evaluation function $f(n) = g(n) + W * h(n)$, where $g(n)$ is the accumulated cost, $h(n)$ the estimated cost of the goal, and $W$ is a constant. Higher values of $W$ are associated with faster plan search, but also with lower plan quality (Korf, 1993). HSP2 evaluates the $h_{add}$ heuristic from the scratch in every new state generated in HSP.

The re-generation of $h_{add}$ is an obvious performance issue, which is a bottle neck of the HSP planners as well as related planners like UNPOP. An attempt to address this problem is HSPr (Bonet and Geffner, 1999), a variant of HSP which uses backward search from the goal rather than forward search from the initial state. The estimates are computed only once from the initial state, and the heuristic function $h_{add}(s)$ is always calculated as sum of costs to achieve goals from the initial state (Bonet and Geffner, 1999). This combination of forward propagation to derive estimations and the backward search for plans is reminiscent of GRAPHPLAN (Blum and Furst, 1995), which is discussed in Sect. 4.2. While HSPr turned out to substantially improve performance in some domains, the new algorithm is inferior to HSP in others (Bonet and Geffner, 1999).

Planning as heuristic search was further advanced by the Fast Forward (FF) planner (Hoffmann, 2001), which was among the winners of the ICP-2000 competition, outperforming HSP2 and others. Like HSP, FF relies on forward search in the state space, guided by a heuristic that estimates goal distances using a relaxed problem. However, FF uses a more sophisticated method of extracting heuristic values, based on a GRAPHPLAN-style algorithm (cf. Sect. 4.2). The number of actions in the relaxed solution is used as a goal distance estimate; among the advantages of GRAPHPLAN-like solution extraction is that it takes positive interactions between facts into account. The estimates are used to guide a novel kind of local search strategy, called *enforced hill-climbing*. In contrast to HSP, which randomly chooses the best successor to each intermediate state, FF evaluates all of a state's successors (and, if necessary, the successors of the successors etc.), looking for a state with better heuristic value than the current state. In short, at each search iteration a breadth first search for a state with strictly better evaluation is performed. This strategy allows the planner to escape plateaus and local minima. A third advantageous feature of FF is its concept of *helpful actions*, i.e. it uses the information from the planning graph to identify at each state those actions that appear most useful in terms of the effects they achieve and it prefers those actions over the operators that seem superfluous (Hoffmann, 2001; Hoffmann and Nebel, 2001).

An extension to FF, called Metric-FF, was presented in (Hoffmann, 2003; Hoffmann, 2002); it handles numerical variables, constraints and effects as captured in PDDL 2.1 level 2. Metric-FF supports numerical state variables which can be used in numerical constraints in preconditions (e.g. $cash > 100$) and in arithmetic operations in effects (e.g. $cash- = 10$).

While the heuristics using a relaxed planning graph has had remarkable success in recent years, in some domains the FF and HSP families of planners perform poorly, because their relaxation method of ignoring negative effects loses too much vital information. A recent heuristic search planner which addresses this problem is Fast Downward (Helmert and Richter, 2004). In contrast to the previous planners like HSP and FF, it does not use a relaxed planning graph but it uses Causal Graph (CG) data structures (Helmert, 2004) instead.

## 4.2 Graph Based Planning

Several planners discussed so far utilize graph structures for the extraction of heuristics. In this section, we will discuss the graph planning framework introduced in (Blum and Furst, 1995), which formalizes the construction, annotation and analysis of a compact structure called Planning Graph. Despite some similarity, Planning Graphs are not space graphs such as those used in UNPOP. In fact, unlike the state-space graph, in which a plan is a path through the graph, in a Planning Graph, a plan is a *flow* in the network flow sense (Blum and Furst, 1997).

A Planning Graph is a directed leveled graph. It consists of two types of nodes, namely *action nodes* and *proposition nodes*. These nodes are arranged in alternating *levels* consisting of proposition nodes followed by layers of action nodes, and so forth. Each level is associated with a time step. The first level of a Planning Graph is a proposition level which consists of one node for each proposition of the initial situation. The second level is an action level which contains all actions whose preconditions are satisfied by the proposition nodes of the first level. The third level is again a proposition level, containing the proposition nodes from the first level and proposition nodes that represent the effects of the actions of the preceding action layer. The construction of the Planning Graph stops when two consecutive propositional layers are identical; it has been shown that this always occurs, guaranteeing the termination of the process. The complexity of creating a Planning Graph is low-order polynomial in the number of actions and propositions (Blum and Furst, 1997).

All actions at some level $i$ are connected to the preconditions at level $i-1$ and its effects at level $i+1$, introducing or negating proposition in $i+1$. For literals that persist from layers $i-1$ to $i+1$ and are not connected to action nodes, persistence action nodes are added. Further, (Blum and Furst, 1995) defines mutual exclusions ("mutex") relations for actions and literals. The possible mutex relations between actions are *inconsistent effects*, where one action negates an effect of the other, *inference*, where one of the effects of one actions is the negation of a precondition of the other and *competing needs*, where one of the preconditions of one action is mutually exclusive with a precondition of the other. A mutual relation holds between two propositions on the same level if one is the negation of the other or if each possible pair of actions that could achieve the two propositions is mutually exclusive (Russel and Norvig, 2002).

The information captured in such a Planning Graph, especially the exclusion relations propagate a variety of intuitively useful facts about the problem. This information can be used by planners to guide their search. The first planner using this technique was GRAPHPLAN, which was introduced in (Blum and Furst, 1995). The GRAPHPLAN algorithm operates in two main steps which alternate within a loop: *graph expansion* and *solution extraction*. Graph expansion grows the Planning Graph as sketched above, until a propositional level is reached where all goal propositions are present with no mutex links between any pair of them. This is a necessary (but insufficient) condition for plan existence. To look for potential plans, the solution extraction phase is then started. The GRAPHPLAN algorithm uses a backward-chaining strategy, using a level-by-level approach in order to make best use of the mutual exclusion constraints (Blum and Furst, 1997). Given a set of goals at a time (level) $t$, GRAPHPLAN aims to determine a set of actions at time $t-1$ which have these goals as effects. At each step, only actions that are not mutually exclusive with existing actions in the plan are considered. On failure, GRAPHPLAN tracks back and tries different action sets. If no plan is found and the Planning Graph is not leveled off yet, then GRAPHPLAN resumes graph expansion until another promising propositional layer is reached. The solution extraction can be formulated as a constraint solving problem[3] (Do and Kambhampati, 2001) or as a search problem (Russel and Norvig, 2002).

GRAPHPLAN's advantages are besides its good performance its theoretical properties such as soundness, completeness, generation of shortest plans and termination on unsolvable problems. However, the original GRAPHPLAN algorithm has some limitations: first, its representation language is restricted to pure STRIPS operators, no conditional or universally quanti-

[3]Since Graph-based planning can be formulated as "constraint satisfaction"-based planning, this terms is frequently used to refer to Graph-based planning. However, it should be pointed out that there exist other constraint-based approaches to planning which do not use any planning graphs, for instance the MOLGEN planner (Stefik, 1981).

fied effects are allowed; and second, the performance can decrease drastically if too much irrelevant information is contained in the specification of a planning task (Nebel et al., 1997).

In (Koehler et al., 1997), an early version of the Ipp planner is presented, which extends Graphplan to handling conditional and universally quantified effects. The authors show that this extension comes with negligible computational overhead and competes well with other planners that support ADL subsets (e.g. UCPOP and Prodigy). In additional work, the RIFO (Removing Irrelevant Operators and Initial Facts from Planning Problems) strategy (Koehler et al., 1997) was added to Ipp, addressing Graphplan's problem with irrelevant information. Further, a Goal Agenda Manager (GAM) to order sets of subgoals and incrementally plan for subproblems (Koehler and Hoffmann, 2000) has been added to Ipp, and more recently to the FF planner.

Another planner based on Graphplan that has evolved over time is Stan (Long and Fox, 1999). It improves Graphplan in several ways. First, Stan performs a number of preprocessing analyses, or STate ANalyses, on the planning domain before planning, using the Type Inference Module (TIM) described in (Fox and Long, 1998). Further, Stan uses an efficient internal representation of preconditions and effects (as bit vectors), which allows for resource-efficient representations of the planning graph (called *spike*) and allows to carry out the mutex-checks between actions and facts using efficient bit manipulating operations[4]. Further, redundant information is avoided in the spike structure, using a technique called *wave front* (Long and Fox, 1999), which results in advantageous space requirements over Graphplan.

Sensory Graphplan SGP (Weld et al., 1998) is an extension to Graphplan that not only supports conditional effects in the way described in (Gazen and Knoblock, 1997), but also deals with uncertain effects (Smith and Weld,

[4]similar techniques are used in Symbolic Model Checking, which is touched in Sect. 5.3

1998) and uncertainty in the initial situation. The idea presented in (Smith and Weld, 1998) is to extend a separate planning graph for each possible world, keeping track of mutual exclusion across the worlds, and then to search backwards for a plan that works in all possible worlds. SGP introduces observational effects of the form *(sense wwf)*, where *wwf* denotes an arbitrary logical expression composed of propositions. Operators in SGP may have zero or more such observational effects, which, when executed at runtime, deliver the truth value of the specified expression *wwf*. To accomplish this extension of GRAPHPLAN, SGP modifies the graph expansion phase, such that it derives knowledge propositions from the sensor definitions and the previous planning-graph proposition layer. In addition, it incorporates a conditioning threat resolution method into the solution extraction phase. Furthermore, SGP generates contingent plans with branches that can rejoin.

## 4.3 Partial Order Refinement Planning

In contrast to the techniques discussed so far, Partial Order Refinement Planners – also called Partial Order Causal Link Planners (POCL) or Partial Order Planners (POP) – formulate the planning problem not as search through the space of *world states*, but rather as a search in the space of *partially-specified plans*, i.e. the nodes of the search space are not states but *plans*, and the arcs between the nodes are not action executions but *plan refinements.*

POCL planners usually produce partially ordered plans, i.e. not all actions have a fixed order in the plan, and a plan may have several different linearizations, which all achieve the identical result.

A partially ordered plan, in older literature also called task network, can be represented as a quadruple $\pi = \langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L} \rangle$, which consists of the following components: $\mathcal{S}$ is a set of plan steps, i.e. instances of operations. $\mathcal{O}$ is a set of ordering constraints. Each ordering constraint is of the form $s_i \prec s_j$, which means that the step $s_i$ must be executed *before* step $s_j$. If the set $\mathcal{S}$ of some plan $\pi$ has at least two steps $s_a$ and $s_b$ where $\mathcal{O}$ neither contains $s_a \prec s_b$ nor $s_a \prec s_b$, then $\pi$ is a *partially ordered* plan. $\mathcal{B}$ is a set of variable binding constraints on the parameters of action instances: Each variable constraint is of the form $var = x$ or $var \neq x$, where *var* is a variable of some plan step and $x$ is either a constant value or a reference to a variable of some other plan step. If only ground plan steps are used, then $\mathcal{B} = \emptyset$. $\mathcal{L}$ is a set of causal links. Causal links are used to keep track why a step was introduced to a plan and to prevent other steps from interfering with that purpose. If a step $s_i$ achieves a proposition $p$ to satisfy a precondition of step $s_j$, the causal link $s_i \xrightarrow{p} s_j$ is added to $\mathcal{L}$.

Further, the following derived sets are considered in partial order planning: $\mathcal{OC}$ is the set of open conditions of a plan. An open condition $\xrightarrow{p} s$ emerges when $p$ is a literal that is part of *Prec(s)* and when there is no causal link $s_x \xrightarrow{p} s$ in $\mathcal{L}$. In other words, open conditions are preconditions of plan steps which have not yet been addressed by the current plan. $\mathcal{UL}$ is the set of unsafe links. A causal link $s_i \xrightarrow{p} s_j$ is called *unsafe* if there exists a step $s_k \in \mathcal{S}$ such that (i) $\neg p$ is part of the effect of $(s_k)$ and (ii) $\mathcal{O}$ is consistent with $\{s_i \prec s_k \prec s_j\}$. In such a case, $s_k$ is said to *threaten* the causal link $s_i \xrightarrow{p} s_j$. The union of a plan's open conditions and unsafe links is called the set $\mathcal{F}$ of *flaws* of $\pi$, i.e. $\mathcal{F}(\pi) = \mathcal{OC}(\pi) \cup \mathcal{UL}(\pi)$. A plan $\pi$ that has no flaws is called *complete*.

An open condition $\xrightarrow{p} s$ can be resolved by introducing or reusing a plan step that has an effect achieving $p$. On the other hand, a *threat* of a causal link $s_i \xrightarrow{p} s_j$ by a step $s_k$ can be possibly resolved either by *promotion*, i.e. by adding an ordering constraint $s_k \prec s_i$ to $\mathcal{O}$ or by *demotion*, i.e. by adding $s_j \prec s_k$ to $\mathcal{O}$. If the planner uses lifted actions, i.e. if it allows action instances with variables in their parameter lists, a threat can also possibly be resolved by *separation*, that is by adding binding constraints such that $p$ and $\neg p$ cannot be unified. The way a planner navigates through plan space, i.e. the strategy it employs to chose the plans to refine and the flaws to remove determines the efficiency of the planner.

The inception of partially ordered planning in 1975 with NOAH (Sacerdoti, ) sparked research and development activities during nearly three decades. In 1977, the NONLIN system (Tate, 1977) was presented, which introduced the concept of causal links. A formal account of partial order planning was given in (Chapman, 1987), which presented the TWEAK system, which could handle conjunctive and disjunctive preconditions as well as conjunctive effects. (Chapman, 1987) also provides proofs of TWEAKS soundness and completeness, whereby the latter is given using the *modal truth criterion* (MTC) to explicitly check that all the safe ground linearizations correspond to solutions. More recent planners, however, depend on *protection strategies* and *conflict resolution* to indirectly guarantee the safety and necessary correctness of the solution: SNLP (McAllester and Rosenblitt, 1991) introduces the notion of threats and safety conditions, and UCPOP (Penberthy and Weld, 1992) extends the complexity of domain and problem descriptions to actions that have conditional effects and universally quantified preconditions and effects and universally quantified goals. Much work has followed to scale up SNLP and UCPOP, most importantly involving heuristics for efficient flaw selection which was improved in (Peot and Smith, 1993; Williamson and Hanks, 1996; Schubert and Gerevini, 1995; Pollack et al., 1997). Despite those gradual improvements, partial order planning was outperformed by the new Planning Graph-, SAT- and heuristic state space planners which have emerged in the second half of the 1990ies.

In recent years (since around 2001) several promising attempts have been carried out to reclaim some of the repudation of POCL planning: in (Nguyen and Kambhampati, 2001) it was noted that the techniques responsible for the efficiency of GRAPHPLAN and heuristic state planners can be adapted to dramatically improve the efficiency of partial order planning. (Nguyen and Kambhampati, 2001) introduce REPOP, a POP implementation that incorporates several enhancements: it uses a Planning Graph to compute an estimation of the cost of achieving a set of (sub-) goals. Further, it uses a novel technique of handling unsafe links: An unsafe link $a_i \xrightarrow{p} a_j$ that is in conflict with an action $a_k$ is not automatically solved by promotion or demotion, which would result in new partial plans, eventually blowing up the plan space and decreasing performance; instead, RePOP uses *disjunctive constraint handling*, proposed in (Kambhampati and Yang, 1996), which is to resolve the unsafe link by posting a disjunctive ordering constraint that captures the promotion and demotion possibilities, and incrementally simplifies these constraints by propagation techniques, which results in the detection of many failing plans before they get selected for refinement. Further, RePOP improves the consistency enforcement of partial order planning. POP considers the causal link $a_i \xrightarrow{p} a_j$ only threatened by an action $a$, if it has an effect $\neg p$. Often $a$ might have an effect $q$ such that no legal state can have $p$ and $q$ true together. In order to detect such implicit conflicts, information about reachable states is required. Again, a Planning Graph is employed to generate this information. The reachable states are then contrasted with pre- and post-*cutsets*, which are unions of pre- and post-conditions derived from the chains of plan steps defined by causal link and ordering constraints. If these cutsets violate the properties of the reachable states, i.e. if a mutex is detected (cf. Sect. 4.2), then the partial plan is discarded. The result of these enhancements is that RePOP is able to demonstrate equal and sometimes better performance than the CSP and state based planners it has borrowed the powerful conflict detection techniques from. At the same time, it is able to generate more compact solutions in many cases and it retained the openness and flexibility of the POP framework, which is considered one of the advantages of that framework (Smith et al., 2000).

Another recent advancement of partially ordered planning was achieved by the Versatile Heuristic Partial Order Planner (VHPOP), presented in (Younes and Simmons, 2002; Younes and Simmons, 2003), which competed successfully at the 3rd International Planning Competition IPC-3.

Like SNLP and UCPOP, VHPOP uses the $A*$ algorithm (Hart et al., 1968) to search through the plan space. The $A*$ algorithm requires a search node evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of getting to $n$ from the start node (i.e. the initial plan) and $h(n)$ is the estimated remaining cost of reaching the goal node (i.e. the complete plan). To encourage search for minimal plans, the cost of a plan is the number of actions in it; while SNLP and UCPOP use the number of open flaws $\mathcal{F}(\pi)$ of a plan to give an estimation of $h(\pi)$, the VHPOP planner adapts the additive heuristic $h_{add}$ of HSP (cf. Sect. 4.1) to achieve a better informed heuristic, which takes into account positive interactions between goals. Like the state space planner FF, VHPOP utilizes a relaxed Planning Graph to extract the data for $h_{add}$.

While these heuristics inform *plan selection*, VHPOP also provides powerful heuristics for *flaw selection*: Besides implementations of existing strategies such as DUnf and DSep (Peot and Smith, 1993), LCFR (Joslin and Pollack, 1994), and ZLFIO (Schubert and Gerevini, 1995), VHPOP introduces novel flaw selection strategies: *Static-first* deals efficiently with static open conditions, *LCFF-Loc* allows for local flaw selection, which makes the planner less sensitive to precondition order in operator specifications, and several conflict-driven flaw section strategies are introduced which build on the assumption that those open conditions which would be threatened when closed, should be treated with higher priority (Younes and Simmons, 2003).

Furthermore, VHPOP extends the POP framework to handle durative actions as specified by PDDL 2.1 level 3, attaching temporal annotations to open conditions which tell the planner at which time step the condition must hold.

## 4.4 Planning as Satisfiability

The idea behind the *planning as satisfiability*-approach is to express the planning problem as a reasoning problem for which powerful problem solving algorithms exist.

### 4.4.1 Planning as Propositional Satisfiability

The logical approach to planning has traditionally been *deduction* (e.g. (Green, 1969; McCarthy and Hayes, 1987; Rosenschein, 1990)), that is, to find a proof that the initial conditions together with the domain axioms (which define the semantics of the operators) and some sequence of actions imply the goal situation (expressed as logical formula). The proof for such a theorem is any valid instantiation of the logical theorem, and the action sequences can be extracted from such an instantiation.

However, in (Kautz and Selman, 1992) planning through *satisfiability* checking was presented. In that approach, a planning problem is not a theorem to be proved, instead it is formulated as a set of axioms with the property that *any* model of the axioms correspond to a valid plan. This property is achieved by crafting axioms that rule out unintended (anomalous) models, for instance axioms are needed to explicitly rule out the possibility of executing an action while its precondition is not fulfilled.

Further, the axioms do not contain quantification or terms, and all predicates have a trailing argument that takes a time step. For instance[5], in the well known blocks world (Gupta and Nau, 1992), the planning problem of achieving $on(B, A)$ from an initial situation $on(A, B) \wedge on(B, Table)$ would be expressed as:

$$on(A, B, 1) \wedge on(B, Table, 1) \wedge clear(A, 1) \wedge on(B, A, 3)$$

Further, the semantics (preconditions and effects) of the *move* operator would be formalized as:

$$\forall x, y, z, i.move(x, y, z, i) \supset (clear(x, i) \wedge clear(z, i) \wedge on(x, y, i))$$

[5]The example was taken from (Kautz and Selman, 1992), which lists all axioms of the example; a complete account of formalizing planning problems in SAT can be found in (Kautz et al., 1996).

Additional axioms are needed to make sure that exactly one action is taken at one time step. In this example, the only model that satisfies the axiomatized planning problem is:

$$\{move(A, B, Table, 1), move(B, Table, A, 2)\}$$

which is the intended model and which can serve as a plan for an agent. The models can be constructed using satisfiability decision procedures such as the Davis-Putnam procedure or stochastic procedures such as GSAT (Selman et al., 1992); another example is WalkSAT, also called WSAT (Selman et al., 1993).

This approach to planning turned out to be highly competitive (Kautz and Selman, 1996). Planning procedures based on this techniques are SATPLAN (Kautz and Selman, 1992) and the successor BLACKBOX (Kautz and Selman, 1998a) which combines SATPLAN with ideas from GRAPHPLAN. Both systems competed well in the international planning competitions.

A similar approach is taken by the LGP system (Gerevini and Serina, 2002) and its successor LGP-td (Gerevini et al., 2004). Both are based on WalkSAT, but incorporate a best-first search algorithm and use a Planning Graph for search heuristics.

Beside the good performance, SAT based planning has another advantage: since states are modeled explicitly in this framework (the trailing arguments of the atoms), it is easy to formulate requirements on states, as discussed in (Huang et al., 1999; Kautz and Selman, 1998b). This, in turn is useful for expressing the complex goals discussed in Sect. 3.3.

### 4.4.2 Planning as Description Logic Satisfiability

Another approach that presents the planning task as a logical satisfiability problem is presented in (Berardi et al., 2003). More precisely, the article concentrates on the problem of automatic Web service composition. In this approach the "target logic" is not Propositional Logic, but the Description Logic $\mathcal{ALU}$.

The approach defines a *community* of Web services, which is characterized by a common set of actions, called the alphabet of the community and a set of Web services specified in terms of the common set of actions. The interaction protocols a service offers are expressed as *execution trees*, where each node represents a possible state in the client-server interplay and each edge represents an action invoked by the client following the protocol. The Web service composition problem is now to identify an execution tree composed of the actions of the services in the community, that corresponds with a given desired execution tree.

The task of constructing such a tree is reminiscent of the task of producing a model of a description logic concept to prove its satisfiability (or its unsatisfiability): to exploit this property, the Web service domain and planning problem are transformed from a situation calculus representation to $\mathcal{ALU}$, and then any of the highly efficient tableau-based Description Logic reasoners (such as (Volker Haarslev, 2001; Horrocks, 1999)) can be used to generate the tree model of the satisfiability check, from which the synthesized process (if it exists) can be directly extracted.

### 4.4.3 Planning as Petri-Net Reachability

Another work that starts with a situation calculus based axiomatization of the planning problem and then transforms it to fit into a well-known formal framework was presented in (Narayanan and McIlraith, 2002): This article suggests that Web service planning tasks can be carried out under the notion of *reachability analysis* of Petri nets (Petri, 1962). The idea is to create a Petri net based on atomic Web services that represents all possible combinations of atomic operations and to specify the desired goal as a state of this Petri net (i.e. as a configuration of tokens). Standard Petri net techniques, among them satisfiability checking, can then be used to determine if this goal state is reachable. These techniques can also be used to validate whether a plan is well-formed (Narayanan and McIlraith, 2002).

## 4.5 Planning as Logic Programming

Another approach that proposes a way to encode the action laws of a planning domain as a logical representation that is amendable to formal reasoning methods is the Planning as *Logic Programming* approach.

A Logic program is composed by a set of Horn clauses of the form $A \leftarrow B_1, ..., B_n$. Each such Horn clause can also be interpreted as a disjunction of literals with at most one positive literal, i.e. $A \vee \neg B_1 \vee ... \vee \neg B_n$. Negativity in Logic programs is usually expressed as *negation-as-failure* (NAF), which makes them nonmonotonic.

The relation between logic programming (LP) and planning, as well as the encoding of planning problems as Logic programs is extensively studied in the literature, e.g. in (Gelfond and Lifschitz, 1993; Turner, 1997; Lifschitz, 1999).

While the direct application of deductive reasoning, such as Prolog's SLD would appear self-evident, much of the progress of the Planning as LP-approach has been achieved using alternative methods, inspired by (Subrahmanian and Zaniolo, 1995). (Subrahmanian and Zaniolo, 1995) show that planning problems can be converted to Logic programs in linear time such that a given goal $G$ is achievable in the planning domain if and only if a related goal $G*$ is true in some *stable model* of the logic program obtained by the transformation; the goal $G*$ can be obtained in linear time as well (Subrahmanian and Zaniolo, 1995). These stable models can be efficiently generated by computing the *answer sets* of the logic program, as implemented in tools like SMODELS (Niemelae and Simons, 1997). (Dimopoulos et al., 1997) report empirical tests of this approach, and suggest that, given a proper encoding of the Logic programs, the performance can keep up with the performance of general-purpose planning algorithms such as GRAPHPLAN or SATPLAN.

Other applications of Logic programming have been Reiter's implementation of Golog and regression for situation calculus (Reiter, 2001). Also mentioned should be (Shanahan, 2000) which presents a Logic programming encoding of the event calculus. In the realm of Web service composition, the SWORD toolkit was presented in (Ponnekanti and Fox, 2002), which uses Prolog to reason about information providing services, whereby plans are extracted directly from the execution trace of the Prolog interpreter.

# 5 Planning with Control Knowledge

Although the performance of many of the planners described in the last sections is quite impressive, there is a belief among many researchers, that it is necessary to provide the planing agent with domain- or task dependent control knowledge in order to achieve good performance in real world domains.

In the following sections, we will briefly review planning techniques that allow to incorporate and exploit domain or task-dependent control knowledge in one way or the other.

## 5.1 Hierarchical Task Network Planning

Hierarchical Task Network (HTN) planning was first introduced in the early ABSTRIPS (Sacerdoti, 1973) planning system, followed by NOAH and several other planners; it was given a formal semantics in (Erol et al., 1994b; Erol et al., 1994a).

HTN planning provides hierarchical abstraction, a powerful strategy to deal with the complexity of large and complicated real world planning domains. Like other planning paradigms, HTN planning assumes a set of operators that achieve certain defined effects when its preconditions hold directly before its execution. However, in addition to operations (which are called *primitive tasks* in HTN planning), HTN planning also supports a set of *methods*, where each is a prescription for how to decompose some task into some set of subtasks. Such method descriptions represent common *domain knowledge*, or if viewed from the planner's perspective, represent *plan fragments*.

According to the definition in (Erol et al., 1994b), there are three types of goals in HTN planning: (i) goal tasks, which are desired properties of the final state, just like in most other planning paradigms, (ii) the already mentioned primitive tasks, and (iii) compound tasks that denote desired changes that involve several goal tasks and primitive tasks.

A variant of HTN planning which received much attention recently is *ordered task decomposition* planning, where the agent plans for tasks in the same order that they will be executed, which reduces the complexity of the planning problem greatly. Planners based on that principle, like SHOP (Simple Hierarchical Ordered Planner) (Nau et al., 1999) accept goals as task lists, where compound tasks may consist of compound tasks or primitive tasks; goal tasks are not supported. Hence, ordered task decomposition system do not plan to achieve a defined (declarative) goal, but rather to carry out a given (complex or primitive) task.

Such a HTN based planning system decomposes the desired task into a set of sub-tasks, and these tasks into another set of sub-tasks (and so forth), until the resulting set of tasks consists only of primitive tasks, which can be executed directly by invoking atomic operations. During each round of task decomposition, it is tested whether certain given conditions are violated. The planning problem is successfully solved if the desired complex task is decomposed into a set of primitive tasks without violating any of the given conditions.

An approach of using HTN planning in the realm of Web Services was proposed in (Hendler et al., 2003), facilitating the SHOP2 system (Nau et al., 2003), which belongs to the family of *ordered task decomposition* planners we described above. The papers (Hendler et al., 2003; Wu et al., 2003) present a transformation method of OWL-S processes into a hierarchical task network. OWL-S processes are, like HTN task networks, pre-defined descriptions of actions to be carried out to get a certain task done, which makes the transformation rather natural. The advantage of the approach is its ability to deal with very large problem domains; however, the need to explicitly provide the planner with a task it needs to accomplish may be seen as a disadvantage, since this requires descriptions that may not always be available in dynamic environments.

## 5.2 High-level Program Execution

In the classical approaches to planning, a plan is synthesized given a domain description and a planning goal, where the planner has to search a – usually very large – space of possibilities to identify a proper solution to the planning problem. An alternative approach is *high-level program execution*; the idea behind this approach is that, instead of searching for a sequence of actions that satisfies some declarative goal, the task is to identify a sequence of actions which constitutes a *legal execution* of a given high level program. As in planning, it is necessary to reason about the preconditions and effects of the domain's operators to find out which actions can be applied in detail. If the high level program is formulated in detail and is formulated deterministically, then not much reasoning is left to be carried out; otherwise, when the programm is formulated not in detail and if it makes use of nondeterministic control constructs, then the search task begins to resemble traditional planning (Giacomo et al., 2000).

The Golog (alGOL in LOGic) (Levesque et al., 1997) language is such a high-level programming language, and it is particularly designed for the specification and execution of complex actions in dynamic domains. Further, it is logic-based, which means that Golog program are amendable to formal verification procedures and Golog based planning problems can be carried out by logic tools such as theorem provers.

Golog is based on the situation calculus (cf. Sect. 3.1), which was introduced by (McCarthy, 1963) and since then is often used as a means for providing a formal account of dynamic systems. While early treatments of the situation calculus identify situations with states, i.e. a description of the universe at an instant of time (McCarthy and Hayes, 1969), more recent formal treatment (Levesque et al., 1998; Pirri and Reiter, 1999) of the situation calculus identifies situations with world histories. All changes to the world are results of named actions. A situation is a possible world history, resulting from a sequence of actions, and it is expressed as a first order term. The constant $S_0$ denotes the *initial situation*, i.e. the empty sequence of actions. The function $do(\alpha, s)$ denotes the situation that results from executing action $\alpha$ in situation $s$. Actions are written as functions and may be parameterized. For example, the function $paint(?o, ?c)$ might stand for painting an object $?o$ with color $?c$, in which case $do(paint(Door, Red), s)$ would denote the situation resulting from painting the *Door* with red color. The expression $do(putDown(Red), do(paint(Door, Red), do(pickUp(Red), S_0)))$ denotes the situation resulting from executing *pickUp(Red)* in situation $S_0$, then executing *paint(Door, Red)*, followed by *putDown(Red)*.

Golog builds on top of the situation calculus by providing a set of extra-logical constructs which serve as *abbreviations* for logical expressions in the language of the situation calculus. The abbreviation $Do(\delta, s, s')$, *macro-expands* into a situation calculus formula that says that it is possible to reach situation $s'$ from situation $s$ by executing a sequence of actions as specified by $\delta$, which is a complex action expression. Golog provides the following macro-expandable language constructs: primitive actions, test actions, sequence, nondeterministic choice of two actions, nondeterministic choice of action arguments and nondeterministic iteration (while loops). Golog also allows for the definition of (possibly recursive) procedures (Reiter, 2001).

Given a situation calculus-based domain axiomatization *Axioms*, an initial situation $S_0$ and a Golog program $\delta$, the planning (i.e. program execution) task can be expressed as the following theorem proving task (Reiter, 2001):

$$Axioms \vdash (\exists s) Do(\delta, S_0, s)$$

In other words, the planner has to identify a situation $s$ for which the macro-expansion is provable from the axioms. The instance of $s$ is obtained as a side effect of the proof, and from this instance a sequence of actions $\vec{a}$ can be extracted such that $Axioms \models Do(\delta, S_0, do(\vec{a}, s))$ holds, where $\vec{a}$ abbreviates $do(a_n, do(a_{n-1}, ..., do(a_1, S_0)))$. A Prolog-based implementation of a Golog interpreter is presented in (Levesque et al., 1997).

A variant of Golog capable of dealing with concurrency is ConGolog (Concurrent Golog) (Giacomo et al., 2000): it allows concurrent processes, whereby the concurrency is established by interleaving the atomic actions in the component processes; it also supports interrupts (e.g. to handle the situation when an alarm button is hit in an elevator) and exogenous actions, i.e. actions that may occur in parallel to the program but are not under control of the interpreter. A Prolog-based interpreter for ConGolog is presented in (Giacomo et al., 2000).

In (McIlraith and Son, 2001; McIlraith and Son, 2002) a modified version of ConGolog is applied to the problem of Web Service composition. The ConGolog interpreter is extended with the ability to include customized user constraints, a more flexible variant of Golog's sequence construct and the ability to implement sensing actions as external function calls. In (Narayanan and McIlraith, 2002), a formal tranformation from OWL-S to situation calculus and Golog is given. In this context, OWL-S processes can serve as specification of desired processes to be carried out as well as a description of the atomic and complex actions offerered by Web services. The Web service composition problem would then be to find an execution of a Golog program that does satisfy the properties defined in the goal.

## 5.3 Planning As Model Checking

The planning as model checking approach was first proposed in (Cimatti et al., 1997; Giunchiglia and Traverso, 1999), which formulates the planning problem as semantic model checking problem. Model checking is a formal verification technique, which allows to determine whether a property holds in a certain system formalized as a finite state model. This technique is used for the verification of hardware and software systems, to achieve a formal account of the system's behavior, e.g. that a system does never reach a certain unwanted state (safety), or that it is guaranteed to reach a certain state at some point (liveness). For example, the SPIN model-checker was used to verify the multi-threaded plan execution module in NASA's DEEP SPACE 1 mission and discovered five previously unknown concurrency errors (Havelund et al., 2001). A more general, detailed discussion of model checking for reasoning over systems can be found in (Halpern and Vardi, 1991).

Planning by model checking (PBM) is semantically well founded and is capable of dealing with nondeterminism, partial observability and extended goals. In PBM, the planning domain is formalized as a nondeterministic state-transition system, where an action is a transition that may bring the system from one state to a set of possible successor states. As in other planning approaches, planning goals may be expressed as constraints about a desired goal state; additionally, goals may be extended by statements about properties about the *plan itself*, e.g. by CTL (Computation Tree Logic) temporal formulas (Emerson, 1990) or using the recently proposed EAGLE (Lago et al., 2002) language.

The underlying idea of the PBM approach is to generate plans by determining whether the goal formula is true in a model, whereby the model is usually formalized as a Kripke structure. The plans that are generated by PBM are "situated plans" (Georgeff and Lansky, 1990), which are plans that are executed by a reactive agent, which, at each iteration cycle determines the state in which it is situated in, and then applies the action that is foreseen for that state by the plan. To illustrate this more formally, consider a planning domain $\Sigma = (S, A, \gamma)$, where $S$ is a finite set of states, $A$ is a finite set of actions and $\gamma : S \times A \to 2^s$ is the nondeterministic state transition function. Now, a valid PBM-generated plan $\pi$, also called *policy* for this planning domain $\Sigma$, is a set of pairs $(s, a)$ such that $s \in S$ and $a \in A(s)$. It is required that for any state $s$ there is (at most) one action $a$ such that $(s, a) \in \pi$.

Due to the nondeterminism allowed in PBM domains, the definition of a solution differs from the definition of a solution under classical planning assumptions. Depending on the contingency inherent in a solution, it may be either *weak*, *strong* or *strong cyclic*: A weak solution is a solution that may achieve the goal but does not guarantee to do so. A strong solution is guaranteed to achieve the goal regardless of the nondeterministic nature of the domain. A strong cyclic solution is one which guarantees to achieve the goal, if *fairness assumptions* are granted which state that the loops the solution foresees will eventually terminate (Giunchiglia and Traverso, 1999).

Analogously, several algorithms have been proposed, capable of either constructing weak (e.g. (Cimatti et al., 1997)), strong (e.g. (Daniele et al., 2000)) or strong cyclic (e.g. (Cimatti et al., 1998)) solutions. These algorithms have in common that they always terminate.

Since real-world problems involve models that may contain very large numbers of states, practical implementations of these algorithm usually resort to *Symbolic Model Checking* (Burch et al., 1990). In Symbolic Model Checking the sets of possible states of a Kripke-structure and the transition relations between states are represented *symbolically*, usually using vectors of variables that represent the truth value of propositions in states, allowing for more concise, less redundant representation of states and for efficient application of set-theoretic and logical operations. Planning is performed by searching through sets of states, rather then individual states, and the plans themselves are represented as formulas. The practical implementation of the representation and the reasoning techniques of Symbolic Model Checking is often carried out using Binary Decision Diagrams (BDDs) (Bryant, 1986).

One implementation of the Planning for Model Checking approach is MIPS (Edelkamp and Helmert, 2000), which is based on BDDs. The main strength of MIPS is its precompilation phase, which identifies implicit domain knowledge, e.g. state invariants, which, when properly encoded can lead to a reduction of the state description length. Further, MIPS implements numerous novel techniques to increase the efficiency of BDD traversal (Edelkamp and Helmert, 2000).

Other PBM implementations for deterministic domains are Proplan (Fourman, 2000) and BDDPlan (Hölldobler and Störr, 2000); however, these lack a MIPS-like pre-compilation phase and therefore do not reach the high performance of MIPS in larger domains.

While MIPS, Proplan and BDDPLan are designed for *deterministic* domains, systems like MBP (Model Based Planner) (Bertoli et al., 2001) and UMOP (Universal Multi-agent Obdd-based Planner) (Jensen and Veloso, 2000) have been designed to leverage a key advantage of model checking, which is to deal with *nondeterministic* environments: MBP can deal with uncertainty on the initial situation, on the action effects and on the state in which the actions are executed. It uses its own action description language NuPDDL, a variant of PDDL 2.1 which can express uncertainty in the initial state, nondeterministic action effects and non-perfect sensing actions.

Similarly, UMOP uses its own domain description language NADL (Nondeterministic Agent Domain Language); NADL problem and domain specifications are translated into symbolic Kripke structures, represented by OBDDs (Bryant, 1986).

## 5.4 Temporal Planning

The term "temporal planning" does not necessarily refer to a particular planning technique in the narrow sense, it rather refers to the ability of planners to deal with *temporal aspects* of planning domains and problems. Most planning paradigms have been extended in some way to support some temporal aspects of planning. Examples of such temporal aspects are:

- Durative actions: In classical planning approaches actions are usually formalized as having no temporal extension. However, in many domains, the duration of actions play a role. As a consequence the exact timing of effects and time efficient plans are of interest to the planner.

- Validity intervals of propositions: in classical planning, a proposition remains unchanged until it is explicitly changed by the agent using an operator. In the real world, many fluents are dependent on time; for instance the permission to access a Web service may be valid only during a defined temporal interval.

- Concurrent actions: classical planning usually assumes that only one action is executed at once. POP and its partially ordered plans appear like an exception, but it is not that the temporal concurrency of actions in such plans is deliberatively sought; it just means that these actions are independent and that no constraints excluding their concurrency have been identified. Some problems, however, *require* that two actions must be executed at the same time because a sequential operation would not yield the desired result.

- Specification of temporally extended goals (Bacchus and Kabanza, 1996), which do not only express classical goals of achieving some final state, but also express a set of acceptable *sequences of states*. Temporally extended goals may also extend temporal deadlines, safety and maintenance goals (Weld and Etzioni, 1994; Penberthy and Weld, 1992).

The problem of *durative actions* was already addressed by early planners such as the partial order planner NONLIN (Tate, 1977) and the hierarchical SIPE (Wilkins, 1988) planner and is also addressed by more recent planners like VHPOP(Younes and Simmons, 2003), LPG (Gerevini and Serina, 2002) and MIPS (Edelkamp and Helmert, 2000).

*Temporally extended goals* have been addressed by the TLPlan (Bacchus and Kabanza, 1995; Bacchus and Ady, 2001) system, which supports goals specified in an extended version of the Metric Interval Temporal Logic (MITL) (Alur et al., 1996). TLPlan is based on a forward-chaining planning engine. Usually, forward-chaining planners (or state-space planners in general) evaluate the contribution a state makes towards the desired goal by determining its heuristic value (cf. Sect. 4.1). TPLan takes a different approach. It treats each state as a database which it checks against an inverse formulation of the goal formula, and it prunes each state that satisfies such a formula, because this means that it violates a property of the desired goal. It should be noted that the notion of a temporally extended goal formula can be extended to the notion of *domain control knowledge*, which encodes hints to the planners by specifying desired or undesired properties of the states it is supposed to identify.

A successor of TLPlan is TALPlanner (Kvarnstrom and Haslum, 2001), which is, like TLPlan, based on forward search, but uses the TAL language instead of MITL to specify the planning goals and domain control knowledge. TAL is a narrative-based linear metric time logic used for reasoning about action and change in incompletely specified dynamic environments. A TAL goal (or control) formula is input to TALplanner which then generates a solution that entails the goal (or control) formulas.

## 6 Discussion and Outlook

We will now review the most important requirements we previously discussed and we will contrast them with a selection of state-of-the art planning systems. The core requirements we identified for our problem scenarios (cf. Sect. 2) are:

1. The *domain complexity* should support a significant subset of ADL: for instance, universally quantified effects are needed to describe web services that deal with multiple objects (e.g. an operation that removes all items from a virtual shopping cart). Explicit markup of sensing actions and nondeterministic service results is desirable as well.

2. Support for *complex goals* (cf. Sect. 3.3), i.e. "hints" that tell the planner which actions should precede which other actions. This is needed for almost all complex problems, for instance in comparison shopping, where the solution is a sequence of three distinct phases (getting price quotes, making a decision, carrying out the purchase).

3. As already mentioned, the ability to deal with *incomplete information* (cf. Sect. 3.2), for instance to query catalogs in the Web shopping domain, is a core requirement for most Web service domains. This, in turn calls for support of *sensing* actions which help the agent to acquire the needed data (e.g. a method that returns a list of products an online retailer sells)

4. Related to the problem of supporting sensing actions is the ability of planners to *dynamically add (or remove) objects* to (or from) the domain, for instance to add knowledge about product information queried from a sensing action, or to properly model a file copy function in the document handling domain.

5. Finally, there is a strong need for dealing with the *nondeterministic behavior* of services: Web service operations may fail during execution time or they may yield unexpected or undesired results; for instance, an airline in the traveling domain may suddenly run out of free seats (possibly violated the IRP assumption), or an image conversion operation in the e-mail replication scenario may fail.

| **Planner** | **Domain complexity** | **Extended Goals** | **Sensing** | **Dynamic Objects** | **Nondeterm. actions** |
|---|---|---|---|---|---|
| FF (state based) | PDDL 2.1 level 1 | No | No | No | No |
| FF-Metric (state based) | PDDL 2.1 level 1, level 2 | No | No | No | No |
| HSP 2.0 (state based) | PDDL/ADL without complex precond./goals | No | No | No | No |
| IPP (Graphplan based) | PDDL/ADL | No | No | No | No |
| SGP (Graphplan based) | PDDL/ADL, without complex negations in precond./goals | No | Support sensing actions that determine the truth value of formulas | No | Produces contingent plans (?) |
| STAN4 (Graphplan, state based) | The STRIPS+Equality subset of PDDL | No | No | No | No |
| VHPOP (POP based) | PDDL 2.1 level 1 and 3 | No | No | No | No |
| BLACKBOX (SAT, Graphplan based) | PDDL/STRIPS with restrictions | No | No | No | No |
| LGP (SAT based) | PDDL 2.1 levels 1,2,3 | No | No | No | No |

Table 1: WSC requirements vs. state-of-the-art neoclassical planners

| **Planner** | **Domain complexity** | **Extended Goals** | **Sensing** | **Dynamic Objects** | **Nondeterm. actions** |
|---|---|---|---|---|---|
| SHOP2 (HTN based) | PDDL/ADL with metrics and time | yes, as HTN Methods | HTN methods may contain explicit sensing actions | ? | HTN Methods can be designed to deal with nondet. actions |
| ConGolog (High level prog.exec.) | Sit.Calc. | yes, as Golog program executions, incl. userdef. constraints | Golog program may contain sensing actions/subgoals | ? | Golog programs can be designed to deal with nondet. actions |
| MIPS (Planning as Mod.Check.) | PDDL/STRIPS + negative preconditions and univ. conditional effects | supports CTL | No | No | No |
| MBP (Planning as Mod.Check.) | PDDL2.1+extensions | temporally extended goals as supported by NuPDDL | Yes | no ? | Yes, can create strong(cyclic) or weak plans |
| TLPlan (Temporal) | ADL + metrics | temporally extended goals as supported by MITL | No | No | No |
| TALPlanner (Temporal) | PDDL 2.1 (or TAL) | TAL narratives | Yes | ? | No (under development?) |

Table 2: WSC requirements vs. planners supporting control knowledge

In the Tables 1 and 2 we contrast a collection of representative planner implementations against our collection of core requirements for WSC problems. The planners listed in Tab. 1 are neoclassical[6] planners based on the techniques we discussed in Sect. 4, and the tools listed in Tab. 2 are implementations of the planning with control knowledge approach discussed in Sect. 5.

Table 1 shows that most of the neoclassical planners we listed allow for domain descriptions of the necessary complexity, i.e. a significant subset of ADL. However, with the exception of SGP, which provides support for incomplete initial states and sensing operations, the rest of the WSC requirements is not supported by any of these planners.

On the other hand, the planners using domain control knowledge offer a much broader support for our requirements: They support complex domains and they allow complex goals as well. For instance, MBP allows to define temporally extended goals and Golog represent goals in a program-like fashion, including branching and iteration. Consequently, nondeterministic domains can be addressed by providing contingency-handling code (e.g. nondeterministic iteration which forces the planner to continue a loop unit the desired effect of some operation is achieved).

Does this mean that the WSC problem is already solved by planners with control knowledge and that neoclassical planners can not be used for the task?

We think this is not the case: While it is apparent that domain knowledge is a key to solving the WSC problem, it is not clear *which* form of domain knowledge is best suited and *how* it should be gathered and encoded. Here, “soft requirements” like the acceptance of the targeted developer communities are relevant as well. The idea of transforming pre-existing process

[6]they are *classical* in the sense that the planning problems are formulated as desired properties on the final state, and they are *neoclassical* in the sense that they use modern planning algorithms.

descriptions (in OWL-S) to domain control information (HTN methods) as exercised by (Hendler et al., 2003) seems a reasonable approach: it does not require the developers to adopt a new logic-based language, but allows them to use wide spread process engineering skills. However, (Hendler et al., 2003) use a restricted variant of HTN planning, i.e. ordered task decomposition planning, which does not support declarative goal tasks. This means that the agent solely depends on the task lists it is given and that is does not attempt to find "creative" solutions on its own. Similarly, sensing actions are not called because the *agent* identifies the need for sensing; instead the sensing action is explicitly predefined in the task list. While this is a useful and pragmatic approach for many domains, we think it would still be interesting to look for alternative approaches that allow for more flexibility on the agent's part, for instance to deal with situations where no predefined strategies exist yet.

Furthermore, the domain-independent neoclassical planners of Tab. 1 are far from being inapplicable to the WSC problem. What is required, however, is a proper architecture that allows for the decomposition of the planning problem into a set of subproblems that match the capabilities of the neoclassical planners.

An example would be an execution monitoring & replanning architecture (e.g. (Haigh, 1998)), where a controller component transforms the problem into a sequence of simpler planning problems and uses the feedback from plan execution to better inform the heuristics of the embedded planner(s). Since the planning problem is split up into a sequence of planning problems, the problem of dynamic object creation and destruction disappears (because the world is re-created at each step) and even the planning for sensing actions becomes possible for classical planners, as informally described in (Peer, 2004a).

Finally, there is a number of unaddressed issues, which, once solved, may turn out to be very helpful regardless of the planning approach taken. One central issue is *automatic domain analysis*; in this survey we discussed several planning implementations which outperformed their direct competition because of *automatically generated* domain knowledge gathered during a preprocessing phase (e.g. STAN, MIPS). Therefore, it appears to be a worthwhile undertaking to identify ways of gathering useful control knowledge from Web service domains. Similarly, the application of *learning techniques* (e.g. based on feedback from earlier runs) may be considered to improve the agent's planning heuristics.

## Acknowledgements

This work has been supported by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 and by the Swiss National Science Foundation under contract 200021-104009. The author would like to thank Juergen Zimmer, Drew McDermott and Maja Vukovic for their comments and valuable feedback.

# References

Alur, R., Feder, T., and Henzinger, T. A. (1996). The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146.

Bacchus, F. and Ady, M. (2001). IJCAI. In *Planning with Resources and Concurrency: A Forward Chaining Approach*, pages 417–424.

Bacchus, F. and Kabanza, F. (1995). Using temporal logic to control search in a forward chaining planner. In *Proceedings of Second International Workshop on Temporal Repre sentation and Reasoning (TIME)*, Melbourne Beach, Florida.

Bacchus, F. and Kabanza, F. (1996). Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222, Portland, Oregon, USA. AAAI Press / The MIT Press.

Berardi, D., Calvanese, D., Giacomo, G. D., Lenzerini, M., and Mecella, M. (2003). e-service composition by description logics based reasoning. In *Description Logics*.

Bertoli, P., Cimatti, A., Dal Lago, U., and Pistore, M. (2003). Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *ICAPS Workshop on PDDL, Informal Proceedings*, pages 15–24.

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

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

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

Bonet, B. and Geffner, H. (1998). HSP - entry at the AIPS-98 planning competition, pittsburgh 6/98.

Bonet, B. and Geffner, H. (1999). Planning as heuristic search: New results. In *ECP*, pages 360–372.

Bonet, B. and Geffner, H. (2001a). Heuristic search planner 2.0. *AI Magazine*, 22(3):77–80.

Bonet, B. and Geffner, H. (2001b). Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. (1990). Symbolic model checking: $10^{20}$ states and beyond. In *Procedings of Symp. Logic in Computer Science*, pages 428–439.

Carman, M., Serafini, L., and Traverso, P. (2003). Web Service Composition as Planning. In *Proceedings of ICAPS'03 Workshop on Planning for Web Services, June, Trento, Italy.*

Castilho, M. A., Gasquet, O., and Herzig, A. (1999). Formalizing action and change in modal logic I: the frame problem. *Journal of Logic and Computation*, 9(5):701–735.

Chapman, D. (1987). Planning for conjunctive goals. *Artif. Intell.*, 32(3):333–377.

Cimatti, A., Giunchiglia, F., Giunchiglia, E., and Traverso, P. (1997). Planning via model checking: A decision procedure for ar. In *ECP '97: Proceedings of the 4th European Conference on Planning*, pages 130–142. Springer-Verlag.

Cimatti, A., Roveri, M., and Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI/IAAI*, pages 875–881.

Daniele, M., Traverso, P., and Vardi, M. Y. (2000). Strong cyclic planning revisited. In *ECP '99: Proceedings of the 5th European Conference on Planning*, pages 35–48. Springer-Verlag.

Dimopoulos, Y., Nebel, B., and Koehler, J. (1997). Encoding planning problems in nonmonotonic logic programs. In *ECP*, pages 169–181.

Do, M. B. and Kambhampati, S. (2001). Planning as constraint satisfaction: solving the planning graph by compiling it into csp. *Artif. Intell.*, 132(2):151–182.

Edelkamp, S. and Helmert, M. (2000). The implementation of Mips.

Emerson, E. A. (1990). Temporal and modal logic. pages 995–1072.

Erol, K., Hendler, J., and Nau, D. S. (1994a). Semantics for HTN planning. Technical Report CS-TR-3239.

Erol, K., Hendler, J. A., and Nau, D. S. (1994b). UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., and Williamson, M. (1992). An approach to planning with incomplete information. In Nebel, Bernhard; Rich, Charles; Swartout, W., editor, *Proceedings of* *the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 115–125, Cambridge, MA, USA. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

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

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *AAAI*, pages 202–206.

Fourman, M. (2000). Propositional planning.

Fox, M. and Long, D. (1998). The automatic inference of state invariants in tim. *Journal of AI Research*, 9:367–421.

Fox, M. and Long, D. (2003). PDDL2.1: An extension to pddl for expressing temporal planning domains.

Gazen, C. and Knoblock, C. (1997). Combining the expressivity of ucpop with the efficiency of graphplan. In *Proceedings of ECP*.

Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17(2/3,4):301–321.

Georgeff, M. P. and Lansky, A. L. (1990). Reactive reasoning and planning. In Allen, J., Hendler, J., and Tate, A., editors, *Readings in Planning*, pages 729–734. Kaufmann, San Mateo, CA.

Gerevini, A., Saetti, A., and Serina, I. (2004). Planning in PDDL2.2 domains with LPG-TD (short paper). In *Proceedings of the International Planning Competition, 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), abstract booklet of the competing planners.*

Gerevini, A. and Serina, I. (2002). Lpg: A planner based on local search for planning graphs with action costs. In *AIPS*, pages 13–22.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL—the planning domain definition language.

Giacomo, G. D. and Lenzerini, M. (1995). PDL-based framework for reasoning about actions. In *AI*IA*, pages 103–114.

Giacomo, G. D., Lesperance, Y., and Levesque, H. J. (2000). Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169.

Giordano, L., Martelli, A., and Schwind, C. (1998). Dealing with concurrent actions in modal action logics. In *European Conference on Artificial Intelligence*, pages 537–541.

Giunchiglia, F. and Traverso, P. (1999). Planning as model checking. In *ECP*, pages 1–20.

Golden, K. (1997). Planning and Knowledge Representation for Softbots.

Green, C. C. (1969). Application of theorem proving to problem solving. In *Proc. of the First International Joint Conference on Artificial Intelligence, Washington, DC*, pages 219–240.

Gupta, N. and Nau, D. S. (1992). On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254.

Haigh, K. Z. (1998). Situation Dependent Learning for Interleaved Planning and Robot Execution.

Halpern, J. Y. and Vardi, M. Y. (1991). Model checking vs. theorem proving: A manifesto. In Allen, J., Fikes, R. E., and Sandewall, E., editors, *Proceedings 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'91*, pages 325–334. Morgan Kaufmann Publishers, San Mateo, CA.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the determination of minimum cost paths. 4(2):100–107.

Havelund, K., Lowry, M. R., and Penix, J. (2001). Formal analysis of a spacecraft controller using SPIN. *Software Engineering*, 27(8):1000–9999.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Zilberstein, S., Koehler, J., and Koenig, S., editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, pages 161–170. AAAI.

Helmert, M. and Richter, S. (2004). Fast Downward - making use of causal dependencies in the problem representation. In *IPC-4, Extended Abstract*.

Hendler, J., Wu, D., Sirin, E., Nau, D., and Parsia, B. (2003). Automatic Web Services Composition Using SHOP2. In *Proceedings of The Second International Semantic Web Conference(ISWC)*.

Hoffmann, J. (2001). Ff: The fast-forward planning system. *The AI Magazine*.

Hoffmann, J. (2002). Extending FF to numerical state variables. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 571–575, Lyon, France.

Hoffmann, J. (2003). The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res. (JAIR)*, 20:291–341.

Hoffmann, J. and Nebel, B. (2001). What makes the difference between HSP and FF? In *IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*.

Hölldobler, S. and Störr, H.-P. (2000). Solving the entailment problem in the fluent calculus using binary decision diagrams. In *Workshop on Model-Theoretic Approaches to Planning at AIPS2000*. Beckenridge. Extended Abstract, see also (?).

Horrocks, I. (1999). FaCT and iFaCT. In Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., and Patel-Schneider, P., editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133–135.

Huang, Y.-C., Selman, B., and Kautz, H. (1999). Control knowledge in planning: benefits and tradeoffs. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 511–517. American Association for Artificial Intelligence.

IBM, Microsoft, and BEA (2002). Business process execution language for web services, version 1.0.

Jensen, R. and Veloso, M. M. (2000). Obdd-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189–226.

Joslin, D. and Pollack, M. E. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the 12th National Conference on Artificial Intelligence AAAI '94, Seattle, WA, USA*, pages 1004–1009. AAAI Press.

Kambhampati, S. and Yang, X. (1996). On the role of disjunctive representations and constraint propagation in refinement planning. In Aiello, L. C., Doyle, J., and Shapiro, S., editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 135–146. Morgan Kaufmann, San Francisco, California.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, pages 359–363. John Wiley & Sons, Inc.

Kautz, H. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In Shrobe, H. and Senator, T., editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California. AAAI Press.

Kautz, H. and Selman, B. (1998a). Blackbox: A new approach to the application of theorem proving to problem solving.

Kautz, H. A., McAllester, D., and Selman, B. (1996). Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pages 374–384.

Kautz, H. A. and Selman, B. (1998b). The role of domain-specific knowledge in the planning as satisfiability framework. In *Artificial Intelligence Planning Systems*, pages 181–189.

Koehler, J. and Hoffmann, J. (2000). On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. volume 12, pages 338–386.

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In *Prof. of the 4th European conference on planning*, pages 273–285.

Korf, P. (1993). Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78.

Kowalski, R. and Sergot, M. (1989). A logic-based calculus of events. pages 23–51.

Kvarnstrom, J. and Haslum, P. (2001). TALPlanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Articial Intelligence.*

Lago, U. D., Pistore, M., and Traverso, P. (2002). Planning with a language for extended goals. In *Eighteenth national conference on Artificial intelligence*, pages 447–454. American Association for Artificial Intelligence.

Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for the situation calculus.

Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83.

Lifschitz, V. (1986). On the semantics of STRIPS. In Georgeff, M. P. and Lansky, A. L., editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 1–9, Timberline, Oregon. Morgan Kaufmann.

Lifschitz, V. (1999). Action languages, answer sets and planning.

Long, D. and Fox, M. (1999). Efficient implementation of the plan graph in stan. *Journal of AI Research*, 10:87–115.

McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 634–639, Anaheim, California, USA. AAAI Press/MIT Press.

McCarthy, J. (1963). Situations, actions and causal laws, Technical Report, Stanford University.

McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*.

McCarthy, J. and Hayes, P. J. (1987). Some philosophical problems from the standpoint of artificial intelligence. pages 26–45.

McDermott, D. (1996). A heuristic estimator for means ends analysis in planning. In Drabble, B., editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 142–149. AAAI Press.

McDermott, D. (2000). The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55.

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

McIlraith, S. and Son, T. (2001). Adapting golog for programming in the semantic web.

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

Moore, R. C. (1985). A formal theory of knowledge and action. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Common Sense World*, pages 319–358.

Narayanan, S. and McIlraith, S. A. (2002). Simulation, verification and automated composition of web services. In *WWW '02: Proceedings of the eleventh international conference on World Wide Web*, pages 77–88. ACM Press.

Nau, D. S., Au, T. C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404.

Nau, D. S., Cao, Y., Lotem, A., and Mu&#241;oz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 968–975. Morgan Kaufmann Publishers Inc.

Nebel, B., Dimopoulos, Y., and Koehler, J. (1997). Ignoring irrelevant facts and operators in plan generation. In *Proceeding of ECP-97*, pages 338–350.

Newell, A. and Simon, H. A. (1963). GPS, a program that simulates human thought.

Nguyen, X. and Kambhampati, S. (2001). Reviving partial order planning. In Nebel, B., editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 459–464. Morgan Kaufmann Publishers.

Niemelae, I. and Simons, P. (1997). Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of Lecture Notes in Artificial Intelligence, pages 420–.429. Springer Verlag.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence.* Morgan Kaufmann, Palo Alto, CA.

Pearl, J. (1985). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, Reading (Massachusetts), USA.

Pednault, E. (1994). ADL and the state-transition model of action. *Journal of Logic and Computation.*

Pednault, E. P. D. (1989). Adl: Exploring the middle ground between strips and the situation calculus. In Brachman, R. J., Levesque, H. J., and Reiter, R., editors, *KR'89: Proc. of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. Kaufmann, San Mateo, CA.

Peer, J. (2004a). 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, St. Malo, France)*, Lecture Notes in Computer Science 3208. Springer Verlag.

Peer, J. (2004b). Sesma semantic service markup: Domains and use cases, http://elektra.mcm.unisg.ch/pbwsc/docs/domains.pdf.

Penberthy, J. S. and Weld, D. S. (1992). Ucpop: A sound, complete, partial order planner for adl. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference KR'92*, pages 103–114.

Peot, M. A. and Smith, D. E. (1993). Threat-removal strategies for partial-order planning. In *Proc. of the Eleventh National Conference on Artificial Intelligence (AAAI Press, ed.)*, pages 492–499.

Petri, C. A. (1962). Kommunikation mit automaten, dissertation, bonn.

Pirri, F. and Reiter, R. (1999). Some contributions to the metatheory of the situation calculus. *J. ACM*, 46(3):325–361.

Pollack, M. E., Joslin, D., and Paolucci, M. (1997). Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research.*

Ponnekanti, S. and Fox, A. (2002). SWORD: A developer toolkit for web service composition. In *Proc. of the 11th International World Wide Web Conference*.

Reiter, R. (2001). On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic (TOCL)*.

Rosenschein, S. J. (1990). Plan synthesis: A logical perspective. In Allen, J., Hendler, J., and Tate, A., editors, *Readings in Planning*, pages 531–536. Kaufmann, San Mateo, CA.

Russel, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice-Hall Inc.

Russel, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach.* Prentice-Hall Inc.

Sacerdoti, E. D. The nonlinear nature of plans. In *Proceedings of the Fourth Joint Conf. on Artificial Intelligence.*

Sacerdoti, E. D. (1973). Planning in a hierarchy of abstraction spaces. In *Third International Joint Conference on Artificial Intelligence, Palo Alto, California.*

Schubert, L. K. and Gerevini, A. (1995). Accelerating partial order planners by improving plan and goal choices. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 442–450, Herndon, Virginia. IEEE Computer Society Press.

Selman, B., Kautz, H. A., and Cohen, B. (1993). Local search strategies for satisfiability testing. In Trick, M. and Johnson, D. S., editors, *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, Providence RI.

Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In Rosenbloom, P. and Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California. AAAI Press.

Shanahan, M. (2000). An abductive event calculus planner. *Journal of Logic Programming*, 44(1-3):207–240.

Sirin, E. and Parsia, B. (2004). Planning for semantic web services (to appear). In *Semantic Web Services Workshop at 3rd International Semantic Web Conference (ISWC2004).*

Smith, D., Frank, J., and J'onsson, A. (2000). Bridging the gap between planning and scheduling.

Smith, D. E. and Weld, D. S. (1998). Conformant graphplan. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 889–896. American Association for Artificial Intelligence.

Srivastava, B. and Koehler, J. (2003). Web Service Composition - Current Solutions and Open Problems. In *Proceedings of ICAPS'03 Workshop on Planning for Web Services, June, Trento, Italy.*

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

Subrahmanian, V. S. and Zaniolo, C. (1995). Relating stable models and AI planning domains. In *International Conference on Logic Programming*, pages 233–247.

Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–893.

Turner, H. (1997). Representing actions in logic programs and default theories. *Journal of Logic Programming*, 31(1-3):245–298.

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

Vukovic, M. and Robinson, P. (2004). Adaptive, planning-based, web service composition for context awareness. In *Second International Conference on Pervasive Computing, Vienna.*

W3C (2002). Web Services Description Language (WSDL) Version 1.2.

Weld, D. and Etzioni, O. (1994). The first law of robotics (a call to arms). In *AAAI'94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, pages 1042–1047. American Association for Artificial Intelligence.

Weld, D. S., Anderson, C. R., and Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *AAAI/IAAI*, pages 897–904.

Wilkins, D. E. (1988). *Practical planning: extending the classical AI planning paradigm.* Morgan Kaufmann Publishers Inc.

Williamson, M. and Hanks, S. (1996). Flaw selection strategies for value-directed planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS'96)*, pages 237–244.

Wu, D., Parsia, B., Sirin, E., Hendler, J., and Nau, D. (2003). Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida.

Younes, H. L. S. and Simmons, R. G. (2002). On the role of ground actions in refinement planning. In Ghallab, M., Hertzberg, J., and Traverso, P., editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling Systems*, pages 54–61. AAAI Press.

Younes, H. L. S. and Simmons, R. G. (2003). VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*.