Planning the Behavior of Dynamical Systems

Nicola Muscettola

CMU-RI-TR-90-10 2

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

April 1990

This work was sponsored in part by the National Aeronautics and Space Administration under contract # NCC 2-531 and by the Robotics Institute, Carnegie Mellon University.

Contents

List of Figures

Abstract

Two important factors hinder our ability to address large planning problems. On one hand, our understanding of planning is not independent from specific planning frameworks. On the other hand, current planning frameworks lack modularity, a key factor for "divide and conquer" approaches to large problems. This paper addresses the formal definition of planning, points out some limitations of the current planning frameworks, and describes a new planning framework that overcomes these limitations. Our formal definition relies on the hypothesis that the problem solver's model of the world is a dynamical system. On this basis, we can clearly separate the knowledge about how the world works from the heuristic knowledge needed to solve problems quickly. Our definition is also independent from any particular planning representation framework. Our analysis of modularity indicates which features can support it in a planning framework. We describe how these features are implemented in the HSTS planning framework, a general purpose facility integrated in the HSTS scheduling architecture. Its effectiveness to address complex "real world" domains has been successfully demonstrated on the problem of building executable observation schedules for the Hubble Space Telescope.

1 Introduction

Two important factors hinder our ability to address large planning problems. On one hand, our understanding of planning is not independent from specific planning frameworks. On the other hand, current planning frameworks lack modularity, a key factor for "divide and conquer" approaches to large problems.

This paper addresses the formal definition of planning, points out some limitations of the current planning frameworks, and describes a new planning framework that overcomes these limitations.

So far, the main efforts to define planning have been devoted to the formal specification of planning programs [4]. This is based on the widespread agreement that the data structures that these programs manipulate indeed represent plans. However, it is also widely understood that other data structures can represent plans [2]. Should we define again what planning is in these different representations? Or should we try to give a representation-independent definition that can be applied to several (and possibly all) representation frameworks?

Another aspect that is sometimes a source of confusion is the relationship between planning and acting. Traditionally, acting has been considered the main justification for developing plans. However, currently there is a consensus regarding the possibility of acting without planning [3]. In the same way, we could observe that there can be planning without acting; for example, we could imagine other outcomes of the battle of Waterloo if Napoleon had made different decisions, or dream about how we would spend the million dollar prize of tomorrow's lottery extraction. The fact is that planning concerns the generation of predictions relative to a model of the world, i.e. a representation of how the world operates. Acting, on the other hand, concerns manipulating the real world, using sensor data to confirm or refute expectations on what should happen, and consequently, adjusting the following actions.

The first part of the paper gives a formal definition of the *planning problem*, one of several problems that a Problem Solver (PS) might address. We will assume that PS possesses an explicit model of the temporal operation of the world and that such a model is independent of the problems that PS can address with it. In particular, we will restrict our attention to those PSs whose model of the world satisfies the axioms of a *dynamical system* [8]. Well-known models, like systems of differential equations and automata, can be regarded as special cases of dynamical systems. We define planning as the selection, from all the possible behaviors of a dynamical system, of some of those that achieve given goals over a given time horizon.

Clearly, the model of the world represents only part of the knowledge that PS must bring to bear during the problem solving process. Heuristic knowledge on the domain will have to be applied in order to focus only on those aspects that are relevant, and therefore, to efficiently obtain a solution. However, these heuristics, which are problem-dependent in nature, must be clearly separated from the description of how the world operates, if we want to build models of potential applicability to other problems. Relying on our formal definition of planning, we will show how the lack of this separation could lead to mistaking problem-specific representations for models of general applicability.

In Section 2 we give the axiomatic definition of dynamical system, while in Section 3 we formalize the planning problem.

In order to represent and solve planning problems, PS needs a representation framework, i.e., a language to describe models of the world and a data structure into which it can build plans. While the definition of the planning problem given in the first part of the paper is independent from the representation framework used, the second part discusses a important

property that planning representation frameworks should possess in order to be applicable to large "real world" domains: *modularity*. This property relates to how a representation framework is able to support the decomposition of a model into elementary components and the determination of the situation of each component at any instant of time. It has already been noted that planning frameworks usually lack modularity [9]; in this paper, we will clearly identify some features that should be present in modular representation frameworks. We then present the HSTS planning framework and show how it supports modularity. The HSTS planning framework is part of HSTS, an integrated planning and scheduling architecture; its effectiveness has been successfully demonstrated on the problem of building executable observation schedules for the Hubble Space Telescope [12].

Section 4 discusses the problem of modularity while Section 5 describes its solution in HSTS.

Throughout the article we will refer to a class of Blocks Worlds (BW) consisting of a **table** supporting a set of blocks, **BLKS**; the blocks can be manipulated either by a robot arm or by a human operator. The arm can find a given block and move it from one place to another, when it receives an appropriate command; the operator can arrange the blocks on the table in any desired configuration.

2 Dynamical Systems

In order to justify intuitively the following discussion, we first sketch the hypothetic cognitive process from which PS's model of the world stems. At different points during its lifetime, PS becomes interested in different portions of the world and in their evolution over time; we will call each of these portions a *dynamical system*. PS identifies which aspects of the system it wants to take into consideration and represents them as the *state* of the system. Usually the system is not isolated from the rest of the world; PS represents its knowledge about the exogenous influences as the *input* to the system. Finally, PS assumes that at any point in time the state is a function of the past history of both input and state.

Describing the world according to input, state, and functional dependency among past state, input and future state is not new to AI [10] and constitutes the basis of the modern theory of control. To characterize the models of the world that result from the previous cognitive process, we use the axiomatization from [8].

Definition 1 (Dynamical System) *A* **dynamical system** $\mathcal{S}$ *is a composite mathematical structure:*

$$\mathcal{S} =< T, X, U, \Omega, \varphi >$$

where T, *the* **time set**, *is an ordered subset of the real numbers* $\mathcal{R}$, X *is the* **state set**, U *is the set of* **input values**, $\Omega = \{u(.) : T \rightarrow U\}$ *is a set of acceptable* **input functions** *and* φ *is the* **state-transition function**:

$$\varphi : T \times T \times X \times \Omega \rightarrow X$$

whose value is the state $x(t) = \varphi(t; \tau, x, u(.)) \in X$ *resulting at time* t *from the initial state* $x = x(\tau) \in X$ *at initial time* $\tau \in T$ *under the action of the input* $u(.) \in \Omega$. *Additional axioms further specify the mathematical structure of* Ω *and* φ *(see [8]).*[1]

[1]The complete axiomatization introduces also the concept of *output* as the only way an external observer can measure the state of the system.

The previous definition accounts both for discrete time and continuous time descriptions of the world, depending on how the set T is extracted from $\mathcal{R}$. Moreover, it does not privilege either input or state in the description of the how the world works; both input and state are described as functions over the same time set T.

The evolution of the system can be experienced by measuring both the input and the state over time, i.e., by perceiving its *behavior*. More formally:

Definition 2 (Behavior) *A pair* $b =< u(.), x(.) >$, *with* $u(.) \in \Omega$ *and* $x(.) : T \rightarrow X$, *is a* **behavior** *of* $\mathcal{S}$ *if* $x(t) = \varphi(t; \tau, x(\tau), u(.))$, *for each* t *and* τ *over which* φ *is defined. The* **behavior space** $\mathcal{B}$ *of* $\mathcal{S}$ *is the set of behaviors obtained by varying* τ, $x(\tau)$ *and* $u(.)$ *in every possible way.*

We now give a complete description of BW that satisfies the axioms of Definition 1. We will use a STRIPS-like representation language.

We choose T_{BW} to be the set of natural numbers, $\mathcal{N}$. At any $t \in \mathcal{N}$, the state of the system $x_{BW}(t) \in X_{BW}$ is a list whose elements can be any of:

1. $\mathsf{ON}(y_1, y_2)$ with $y_1 \in \mathbf{BLKS}$, $y_2 \in \mathbf{BLKS} \cup \{\mathbf{table}\}$ and $y_1 \neq y_2$;
2. $\mathsf{CLEAR}(y_1)$ with $y_1 \in \mathbf{BLKS}$.

To represent a situation physically possible in $\mathcal{S}_{BW}$, $x_{BW}(t)$ must also satisfy the following axioms:

1. $\forall y_1 \in \mathbf{BLKS}$, $x_{BW}(t)$ must contain one and only one $\mathsf{ON}(y_1, y_2)$ (i.e., every block must rest on just one place);
2. $\forall y_2 \in \mathbf{BLKS}$, $x_{BW}(t)$ cannot contain more than one $\mathsf{ON}(y_1, y_2)$ (i.e., every block can support at most one block);
3. $\forall y_2 \in \mathbf{BLKS}$, $x_{BW}(t)$ contains $\mathsf{CLEAR}(y_2)$ if and only if it does not contain any $\mathsf{ON}(y_1, y_2)$ (i.e., a block that does not support anything is clear).

X_{BW} also contains the special list [CRASH] that characterize a situation consequent to the arm moving a supporting block.

The set of possible inputs, U_{BW}, contains $\mathsf{MOVE}(y_1, y_2)$, with $y_1 \in \mathbf{BLKS}$ and $y_2 \in \mathbf{BLKS} \cup \{\mathbf{table}\}$, which represents a command to the robot arm, $\mathsf{RESTORE}(x)$, with $x \in X_{WB}$, accounting for the human operator's intervention, and NO-OP, the lack of any external influence on $\mathcal{S}_{BW}$.

We assume no a priori restrictions on the input; therefore Ω_{BW} is the set of all the infinite sequences of elements from U_{BW}.

Finally, φ_{BW} is defined by the program in Figure 1.

The explicit introduction of a discrete time set does not represent a real novelty with respect to the change-based temporization of classical STRIPS-based descriptions. We could easily obtain it by assuming that the clock is synchronized with each non-empty input value. Formally, this means restricting Ω_{BW} to the set of sequences formed by any combination of indefinite length of MOVEs and RESTOREs, followed only by NO-OPs.

$\mathcal{S}_{BW}$ explicitly represents the undefined state [CRASH]. The rationale behind this is that, although we know that only one of the legal configuration of stacked blocks can result from an "incorrect" MOVE, we have not represented enough information about how the world

$$
\begin{array}{l}
x_{BW}(t+1) := \underline{\texttt{case}}\ u_{BW}(t) \\
\qquad \mathsf{MOVE}(y_1, y_2): \\
\qquad\qquad \underline{\texttt{if}}\ (\mathsf{CLEAR}(y_1) \in x_{BW}(t)) \wedge (\mathsf{CLEAR}(y_2) \in x_{BW}(t)) \\
\qquad\qquad\quad \underline{\texttt{then}}\ x_{BW}(t) - [\mathsf{ON}(y_1, y_3), \mathsf{CLEAR}(y_2)] \\
\qquad\qquad\qquad\qquad + [\mathsf{ON}(y_1, y_2), \mathsf{CLEAR}(y_3)] \\
\qquad\qquad\quad \underline{\texttt{else}}\ [\mathsf{CRASH}]; \\
\qquad \mathsf{RESTORE}(x): \\
\qquad\qquad x; \\
\qquad \mathsf{NO\text{-}OP}: \\
\qquad\qquad x_{BW}(t).
\end{array}
$$

Figure 1: State transition function for $\mathcal{S}_{BW}$.

operates in the model in order to determine, without any sensory input, an accurate estimate of where the blocks will fall. However, we cannot avoid to explicitly include [CRASH] in the dynamical system, if we want to insure the independence of the model of the world from the problems that we can solve on it. In fact, suppose we explicitly represent only the results of "correct" MOVEs (i.e., those moving a non-supporting block on a clear support) [7]. A consequence of this would be the restriction of Ω_{BW} to the sequences that, once plugged in φ_{BW}, produce evolutions of the state without CRASHes; in other words, Ω_{BW} would become a function of φ_{BW}. This could be explained easily if we assumed the existence of an actor that, by knowing φ_{BW} and the goal of excluding CRASHes, appropriately restricts Ω_{BW}. But in this case we would not really be modeling BW but a system whose operation is a function of the solution of planning problems. This is clearly at odds with the requirement of independence of PS's model of the world from the problem to be addressed.

3 The Planning Problem

Having identified a portion of the world and represented it as a dynamical system, PS can now address a planning problem. First of all, PS will focus its attention over a portion of the time line over which the system evolves; it will then consider, of all the possible behaviors, those that, during the time horizon, achieve certain goals. Achieving all the goals is the criterion that PS uses to decide if it has a solution. Moreover, PS will also have preferences that allow it to decide, given any two solutions, which is better. Finally, PS itself will be operating under external constraints (e.g., an actor wanting to use its results) that might limit the amount of time at its disposal [5].

Formally, we define a *planning horizon* H, an interval $[t_s, t_f] \subseteq \mathcal{R}$, with t_s possibly $-\infty$ and t_f possibly $+\infty$. To characterize the focusing of PS to H, we define $\mathcal{B}_H$ as the set obtained by restricting, for each $b \in \mathcal{B}$, each component of b to $T \cap H$. We then define a *goal* $\Gamma(.) : 2^{\mathcal{B}_H} \rightarrow \{\mathrm{T}, \mathrm{F}\}$ and a *preference* $\Pi(.) : 2^{\mathcal{B}_H} \rightarrow \mathcal{R}$. Finally we define a *planning deadline*, $d \in \mathcal{R}^+$ (possibly $+\infty$).

Definition 3 (Planning problem) *Given a dynamical system $\mathcal{S}$, a planning horizon H, a goal Γ, a preference Π and a planning deadline d, find a set $\mathcal{P} \subseteq \mathcal{B}_H$ such that $\Gamma(\mathcal{P}) = \mathrm{T}$ and $\Pi(\mathcal{P})$ has the maximum value among the alternative solutions considered within the deadline d.*

$H_{BW,build(x_s,x_f)} = [0, +\infty]$

$\Gamma_{BW,build(x_s,x_f)}(\mathcal{P}) = [\ \forall b \in \mathcal{P}$
$(u(.)$ does not contain RESTOREs $) \wedge$
$(x(.)$ does not contain CRASHes $) \wedge$
$(x(0) = x_s) \wedge$
$(\exists t_f \geq 0\ (x(t_f) = x_f)\ \wedge\ (\forall t \geq t_f\ u(t) = \text{NO-OP})\ ] \wedge$
$[\ \forall b_1 \in \mathcal{P}\ \forall b_2 \in \mathcal{P}$
(eliminating all the NO-OPs from both $u_{b_1}(.)$ and $u_{b_2}(.)$, we obtain the same sequence of MOVEs)]

$\Pi_{BW,build(x_s,x_f)}(\mathcal{P}) = -$ (number of MOVEs in any $b \in \mathcal{P}$)

$d_{BW,build(x_s,x_f)} = +\infty$

(a)

$\Gamma_{BW,build(x_s,x_f)}(\mathcal{P}) = [\ |\mathcal{P}| = 1\ ] \wedge$
$[\ \forall b \in \mathcal{P}$
$(u(.)$ does not contain RESTOREs $) \wedge$
$(x(.)$ does not contain CRASHes $) \wedge$
$(x(0) = x_s) \wedge$
$(\exists t_f \geq 0\ x(t_f) = x_f)\ \wedge\ (\forall t \geq t_f\ u(t) = \text{NO-OP})\ ]$

(b)

Figure 2: The $build(x_s, x_f)$ problem: (a) complete formulation on $\mathcal{S}_{BW}$; (b) modified $\Gamma_{BW,build(x_s,x_f)}(.)$ for a change-based description of BW.

A *plan* is any $\mathcal{P} \neq \emptyset$. The planning problem has no solution if and only if the empty set is the only one to satisfy Γ.

Going back to $\mathcal{S}_{BW}$ (defined in Section 2), let's consider the problem of generating a sequence of commands for the robot arm to build a given block configuration x_f, starting from a known initial state x_s (problem $build(x_s, x_f)$). Its formulation requires the entities in Figure 2 (a). If we modify Ω_{BW} to obtain a change-based temporization of $\mathcal{S}_{BW}$ (as described in Section 2), we can simplify $\Gamma_{BW,build(x_s,x_f)}$ as shown in Figure 2 (b).

Different PSs can use different representation frameworks (e.g., STRIPS-like, temporal interval-based, etc.) to address the same planning problem and can differ with respect to the heuristic knowledge that they can bring to bear. Their relative performances will depend on both these factors. Notice that the heuristic knowledge that is relevant to effectively solve a planning problem depends on all the entities in Definition 3. For example, given $build(x_s, x_f)$ on $\mathcal{S}_{BW}$, PS could restrict its attention to the portion of φ_{BW} concerning only "correct" MOVE operations; this "heuristic" can be easily proved from the first two constraints on

$u(.)$ and $x(.)$ in $\Gamma_{BW,build_{(x_s,x_f)}}$. Although it is reasonable to expect that any PS with an adequate level of "intelligence" should realize this, this is not a necessary condition to have a PS that solves this planning problem.

Other planning problems could be easily formulated on $\mathcal{S}_{BW}$. For example we might want to build inputs with sequences of MOVEs that every once in a while generate a CRASH, after which the intervention of the human operator RESTOREs the blocks configuration immediately preceding the CRASH. We want to remark that, to solve a planning problem, PS does not need to know how its plans are going to be used. A possible execution environment for a plan that solves the problem just mentioned could consist of an actor that explicitly sends the MOVE and RESTORE commands respectively to the robot arm and to the human operator. But we could also imagine that the actor could send the MOVE commands to the arm but would not have any way to explicitly request RESTOREs from the human operator. This could be the case in a psychological experiment where the goal of the actor is to time the performance of a human in recognizing and recovering from errors in a stream of actions of another agent. The same plan will be used in both cases.

4 The Problem of Modularity

We now turn our attention to the representation frameworks used in planning, and to a property of theirs that is highly desirable: *modularity*. A representation framework is modular if its representation language adequately supports descriptions of systems as collections of interacting components, and its plan data structure allows the determination of the situation of each component at any instant of time. Modularity is one of the keys to "divide and conquer" approaches to modeling and solving planning problems in complex "real world" application domains [12].

In this section we will show that current planning representation frameworks are not modular; consequently, we will identify which structuring features need to be added. Modularization primitives have been introduced in activity based planning frameworks [9]. Instead, our discussion will focus on explicit representations of both state and input, given that in a dynamical system both input and state have equal weight.

We examine the modularity of an interval-based STRIPS representation framework [2]. To conduct our analysis, we consider a restricted *BW* domain with only two blocks, **a** and **b**, and a robot arm, **arm**; we concentrate on the representation of a simple plan to stack **a** on **b**. The relevant operators are given in Figure 3 (a). An operator specifies the facts that have to hold on the state when the operator is applied, and their temporal relations with the action designator; for example, when the first operator is applied, the time interval during which ON(y_1, **table**) holds must overlap that over which MOVE(y_1, y_2) occurs. A database completely describing the resulting behavior is depicted in Figure 3 (b). Notice that a MOVE accounts for the whole control sequence executed by **arm**. With reference to Figure 3 (b), we can identify three distinct phases during MOVE(**a**, **b**); the identification of **a** and approach to it, in $t_1 \leq t < t_2$, the grasping of **a**, moving to **b** and ungrasping of **a**, in $t_2 \leq t < t_3$, and the moving away from **a**, in $t_3 \leq t < t_4$.

To evaluate the modularity of this representation framework, first we need to decide what constitutes a primitive system component. A reasonable initial hypothesis is to use domain objects, i.e. the objects designated by constants in ground facts. Aside from requiring the introduction of an identifier for the robot arm in at least one predicate (for example in MOVE and NO-OP), not all domain objects are system components. Otherwise, if the description

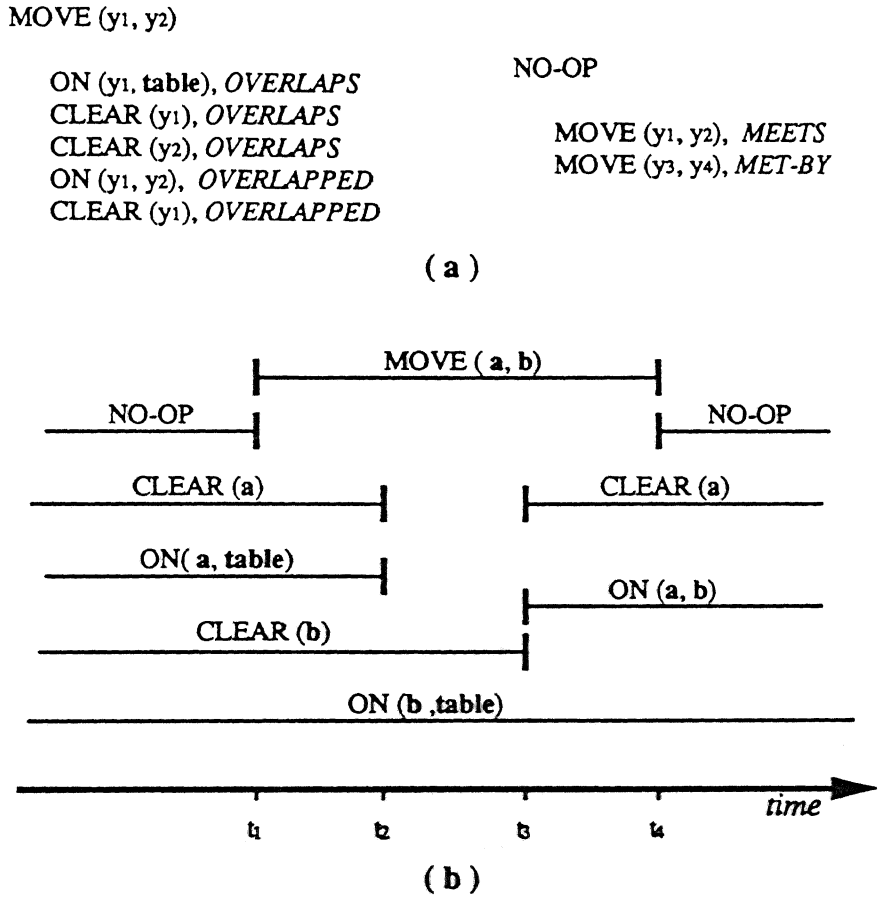

Figure 3: A simple temporal plan. (a) relevant operators; (b) temporal database.

of our domain required predicates like COLOR(a, red) or HEIGHT(b, 3), we would have to consider **red** and **3** as components of the system. Therefore, we need to declare explicitly which domain objects are system components and which are not.

We can now ask how easy it is to obtain the situation of all the system components at any time t. A simple method would be to take all the ground facts that hold in the database at time t and distribute them to the system components that appear as their arguments. So, in the example of Figure 3 (b), before t_1 the situation of **a** is given by CLEAR(a) and ON(a, **table**), while ON(a, b) and ON(b, **table**) account for the situation of **b** after t_4. However the interpretation of MOVE(a, b) is not as straightforward. In fact, while it reasonably represents the situation of **arm** between t_1 and t_4, it directly says something about the situation of **a** only between t_2 and t_3 while it never says anything about the situation of **b**. Otherwise, we would have to assume that the state of an inert object in the physical world changes just as a consequence of an agent taking it into consideration. Therefore, the meaning of a ground fact can change over time with respect to the different system components: we need additional indexing features in the language to resolve this ambiguity.

Finally we could ask if the description of the situation of a system component could not be further decomposed. We could draw a comparison with the way we describe systems in physics; for example, in order to give the behavior of a thermo-electro-mechanical system, we are required to give a continuous function over time for each of the properties of interest of each component of the system (e.g., the position of a body, the temperature of a mass of gas, the electric voltage imposed on a resistance). In our *BW* example, we could be easily convinced that this decomposition in properties is possible. In fact, we can describe the input with the only property "task being executed by **arm**", and the state of each block with the situation of both its top and its bottom.

5 The HSTS Planning Framework

In the previous section, we have identified the following requirements for a representation language that supports a modular description of dynamical systems:

1. explicit declaration of the system components and of their dynamical properties;
2. separate description of the characteristic of each value for each property to which it is associated.

The HSTS planning framework, described below, satisfies the previous requirements. The framework consists of a Domain Description Language (HSTS-DDL), to specify the structure of dynamical systems, and a Temporal Behavior Data Base (HSTS-TBDB), in which it is possible to build plans. For a detailed description of the HSTS planning framework, see [11].

In HSTS-DDL, each system component is represented as a frame; the dynamical properties of the component are slots in the frame. For example, Figure 4 shows the prototypical system components for *BW*.

To specify the system in Section 4, we need one instance of **robot-arm** and two instances of **block**.

Each property can have one and only one value at any instant of time. HSTS-DDL requires the specification of the space of all the possible values that each property can assume. In HSTS-DDL, a value is a tuple; therefore, for each property the space of possible values

{{ robot-arm	{{ human-operator	{{ block
TASK: }}	TASK: }}	TOP: BOTTOM: }}

Figure 4: System components in BW.

< TOP (y_1), MOVE (y_1, y_2) >

< TOP (y_1), CLEAR (y_1) >, *MEETS*
< TASK (robot-arm), MOVE (y_1, y_2) >, *CONTAINS*
< BOTTOM (y_1), MOVE (y_1, y_2) >, *EQUALS*
< TOP (y_1), CLEAR (y_1) >, *MET-BY*

Figure 5: Compatibilities for MOVE of TOP(**block**).

is a set of relations. In BW, for example, the property TOP(y_1), where y_1 is an instance of **block**, can be CLEAR(y_1), ON(y_1, y_2), MOVE(y_1, y_2), where y_2 is either an instance of **block** or the special symbol **table**. Notice that, although **table** is a domain object, it is not considered a system component in our description.

HSTS-DDL requires one to specify a *value descriptor* for each pair $< p, v >$, where p is a property and v is a value. The value descriptor specifies two distinct pieces of information: **1)** duration; **2)** compatibility specification.

The *duration* of a value is the temporal distance between its start event and its end event. Temporal distances are specified as intervals $[d, D]$ ($D \geq d \geq 0$) where d is a lower bound to the distance and D is an upper bound.

The *compatibility specification* consists of sets of compatibilities, not necessarily disjoint. Each *compatibility* is a temporal constraint between a property/value pair $< p', v' >$ and $< p, v >$. The temporal constraints used in HSTS-DDL are equivalent to the temporal relations in [1] but allow also the specification of temporal distances among the extremes of the intervals [6] . The meaning of a compatibility specification for $< p, v >$ is the following: for each behavior b of the system, if the value v appears in b on the property p over an interval of time, then there is a compatibility set in the compatibility specification such that all the compatibilities in the set are satisfied in b.

For example, our specification of BW in HSTS-DDL will contain compatibility specifications relative to the two pairs:

$$< \text{TASK}(\textit{robot-arm}), \text{MOVE}(y_1, y_2) >$$

$$< \text{TASK}(\textit{robot-arm}), \text{NO-OP} >$$

These compatibility specifications will contain compatibility sets corresponding to those in Figure 3 (a). The definition of BW will also contain the compatibility set in Figure 5.

We observe that HSTS-DDL homogeneously treats both input and state properties, in the sense that all values of all properties must have a value descriptor. This satisfies the requirement of equal weight for input and state.

We should not be surprised to find more than one property with the same value at the same time. This is analogous to what happens in continuous descriptions of the behavior of physical systems. There the same real number can describe different things at the same time, i.e., the value of different physical properties in different measuring units.

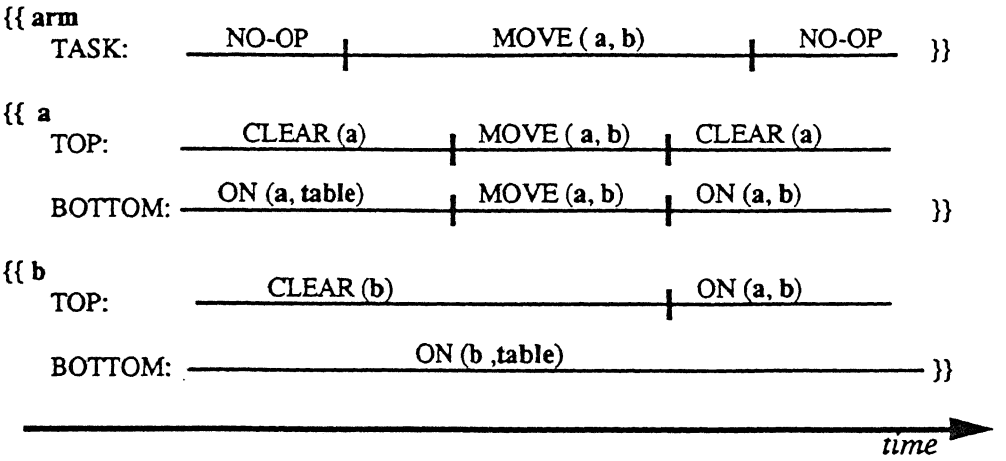

Figure 6: A simple plan in HSTS-TBDB.

The HSTS-TBDB is a constraint-based temporal database that extends the time maps approach [6]. At any point in time, HSTS-TBDB can only represent a set of behaviors of a dynamical system specified in HSTS-DDL. The constraints allow one to leave partially unspecified both the values and the time of occurrence of their start and end events on different segments of the time line. A planner working on HSTS-TBDB can post and refine constraints derived both from the value compatibilities and from the additional conditions added by the planning problem. As an example, Figure 6 depicts the representation in HSTS-TBDB of the same behavior of Figure 3 (b).

Further details on the HSTS planning frameworks can be found in [11].

6 Conclusions

This paper addresses the definition of the planning problem and the representational techniques that we need to solve it.

Our definition of the problem clarifies its mathematical structure and is independent from the representation frameworks over which we can implement planners. Moreover, our approach helps to clearly distinguish the knowledge about how the world works from the heuristic knowledge about how to solve problems quickly.

The HSTS framework is a general purpose facility consisting of HSTS-DDL, a description language to specify dynamical models of the world, and HSTS-TBDB, a constraint-based temporal database in which plans can be built by incremental refinements. The HSTS planning framework solves the problem of modularity, one of the keys to addressing large "real world" applications. The demonstration of the effectiveness of the HSTS approach is an implemented scheduling architecture that has been successfully applied to the generation of executable observation schedules for the Hubble Space Telescope.

Acknowledgments

The author would like to thank Stephen Smith for his helpful comments. Thanks also to Roberto Bisiani, Amedeo Cesta, Daniela D'Aloisi, Robert Frederking and Carlo Tomasi for having read earlier drafts of this paper.

References

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] J.F Allen and J.A. Koomen. Planning using a temporal world model. In *Proceedings of the 8th IJCAI*, pages 741–747, William Kaufmann, 1983.

[3] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[4] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[5] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of AAAI 88*, pages 49–54, Morgan Kaufmann, 1988.

[6] T.L. Dean and D.V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[7] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[8] R.E. Kalman, P.L. Falb, and M.A. Arbib. *Topics in Mathematical System Theory.* McGraw-Hill, 1969.

[9] A.L. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4(1):319–340, 1988.

[10] J. McCarthy and P. Hayes. Some phylosophical problems from the standpoint of artificial intelligence. in *Machine Intelligence 4*, B. Meltzer and D. Michie (editors), pages 463–502. Edinburgh University Press, 1969.

[11] N. Muscettola. *The HSTS Planning Framework.* Technical Report (in preparation), The Robotics Institute, Carnegie Mellon University, 1990.

[12] N. Muscettola, S.F. Smith, G. Amiri, and D. Pathak. *Generating Space Telescope Observation Schedules.* Technical Report CMU-RI-TR-89-28, The Robotics Institute, Carnegie Mellon University, 1989.