Integrating Hierarchical and Conditional Planning techniques into a software design process for Automated Manufacturing[*]

L. Castillo, J. Fdez-Olivares, A. González
Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Granada
{L.Castillo,faro,A.Gonzalez}@decsai.ugr.es
+34.958.240803,+34.958.240805,+34.958.240596

Abstract

This work presents an AI planning application to assist an expert to design control programs in the field of Automated Manufacturing. Authors analyze this domain application, with the aim of understanding the standard procedures carried out by experts in order to solve control software design problems. The analysis results in a set of requirements that have led us to build a hybrid system that integrates POCL, hierarchical and conditional planning techniques. Its advantages and lessons learnend during its development are also shown.

Introduction

Automated Manufacturing involves a wide range of activities, going from plant and product design to operation and control of manufacturing processes. All of these activities share a handicap: they have to reduce costs by saving engineering time, and maintaining the reliability of the engineering process ("do more with less"). Concretely, the design and development of control software for a manufacturing system is a key and fundamental issue for the production of high quality products under an agile, flexible and reliable process. Thus, the former pure economic need leads to a set of requirements that has to be met when a control program is developed: reliability, flexibility and agility for the design process itself, suitability and robustness for its results.

The design of control software for manufacturing systems is also a field of application that is attracting the interest of AI Planning comunity, mainly due to the similarity between the reasoning processes performed by the planning techniques so far developped, and those followed by experts when they solve a design problem. In this sense, AI Planning provides interesting technological foundations to help an expert to design a control program.

But developping an AI Planning application for this domain, aimed to fit all of its requirements, is a complex task. First, due to the complexity of this domain, many planning techniques that are usually developed separatedly (like hierarchical and conditional planning) must be extended and integrated into a single approach. In addition, it is not sufficient to develop only an integrated planning algorithm, many others fundamental issues have to be taken into account, mainly those concerned with how experts solve problems in that domain, what requires a deep understanding of the domain of application.

In this sense, this work presents an AI Planning application that integrates hierarchical and conditional planning techniques for the assisted development of control programs for manufacturing systems. The paper starts with an analysis about the standard procedures carried out by experts when a control program is designed, and the standard formalisms used for its representation. This analysis is used to determine the main requirements to be met by the AI planning application. Then, the resulting AI planning system, which integrates POCL, hierarchical and conditional planning techniques, is described by following an example. In addition, we explain how to integrate this system into a software design environment. Finally, the advantages of using this system as well as the lessons learned during its development are shown.

Domain Application

A Manufacturing system can be thought of as a multiagent system, whose agents exhibit a behaviour globally coordinated in order to carry out a given process on products. This behaviour is guided by a control program that an expert designs by hand. As an example, Figure 1 shows a simple manufacturing system made up of several devices (pumps and

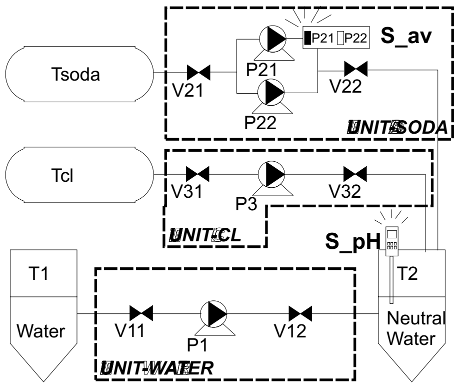

Figure 1: A manufacturing system that neutralizes water.

[*]This work has been supported by the spanish CICYT under project TAP99-0535-C02-01 and TIC2001-4936-E.

valves) which can be conceived as agents which perform a process oriented to neutralize water. The behaviour of these agents is conditioned by the information supplied by sensors SpH and Sav. Thus, if the pH of the water contained in the tank T2 (detected by the sensor SpH) is basic, valves V31 and V32 must be open in order to allow the pump P31 to transport chlorine from the tank TCL to the tank T2. On the other hand, if the pH is acidic, valves V21 and V22 must be open for supplying soda from the tank TSODA to the tank T2. What is more, the sensor Sav determines which pump (P21 or P22) is available to transport soda.

Manufacturing standards for software design

At present, in order to perform a reliable design of a program that controls the operation of a manufacturing system like the one shown, experts follow the ISA-SP88 standard (ISA 1995), a hierarchical methodology for the specification and design of hierarchical and modular control programs. This standard (see Figure 2) allows for a hierarchical specification of the physical, process and control model of a manufacturing system. Thus, starting from a hierarchical physical model and from a process specification (which in this standard is called a *recipe*) at a high level of abstraction, a control engineer performs a hierarchical software engineering process that results in a hierarchical and modular control program at different levels of granularity.

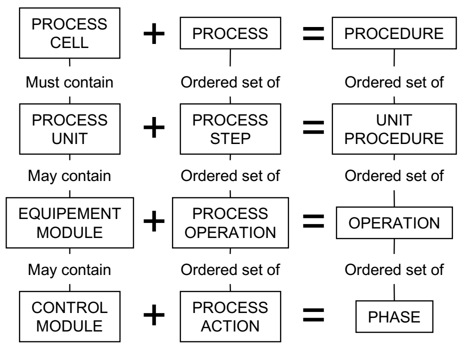

Figure 2: Physical, Process and Control Model of the standard ISA-SP88.

For example, following this standard an expert would define three high level process units,UNIT-WATER,UNIT-SODA and UNIT-CL, which would be composed by lower level devices (see the dotted lines in Figure 1). In addition, the corresponding high-level process for every process unit have to be defined: "*neutralize acidic water*" for UNIT-SODA, "*neutralize basic water*" for UNIT-CL, and "*supply water*" for UNIT-WATER. Thus, once the control procedure, at the highest level of abstraction, for these process units has been designed, the expert continues the design process at a lower level of abstraction, following a stepwise, top-down refinement process.

Finally, experts also use standard tools as GRAFCET (IEC 1988) or Petri Nets (Peterson 1981) for the representation, specification, and validation of the final hierarchical and modular control program so obtained.

This standard methodology and the tools used by control engineers to develop control programs are useful and necessary, but they are not sufficient if we take into account some needs of new generation manufacturing systems as flexibility, quick response and engineering budget reduction (PLANET 2001). From the point of view of these needs, the manual process followed so far has some drawbacks: it is slow, expensive and subject to human errors and, therefore, it becomes necessary to improve this process in order to save more engineering time and, not less important, maintaining the reliability of the process.

A very good direction for covering these needs consists on the improvement of this design process by means of AI Planning techniques (PLANET 1998; PLANET 2001). Although presently it is in an initial state, interesting approaches based on these techniques have been carried out in the last few years (Aylett *et al.* 1997; Castillo, Fdez-Olivares, & González 2000a; Klein, Jonsson, & Backstrom 1998; Nau, Gupta, & Regli 1995; S.Viswanathan *et al.* 1998), which are proving to be very useful, allowing for an error-free, fast and low-cost building process of control programs.

The improvement of the design process of a control program, by means of AI Planning techniques, can be easily understood if we rely on the similarity between the concept of plan and program. Thus, a SP88-based software engineering process, that receives as input the physical model and a high-level process specification of a manufacturing system, is comparable to a planning process which receives as input a knowledge-based description of a manufacturing system, and a high-level problem specification. Therefore, the plans obtained by such a process can be considered as control programs for the manufacturing system described.

AI Planning Application Requirements

The SP88 standard can be seen as a hierarchical problem solving strategy followed by control engineers for the design of a hierarchical control program. Therefore, it is clear that any planning approach aimed to fit present manufacturing standards must be developed under a hierarchical planning paradigm, in which the planning knowledge described in the domain description stage is hierarchically structured. Furthermore, in order to consider the plans so obtained as truly realistic and operational control programs, the planning approach must also represent and manage incomplete knowledge. Thus, the planning techniques applied should lead to the synthesis of plans that incorporate the information supplied by sensors and capable of describing alternative courses of actions to deal with foreseeable contingencies (that is, closed-loop, robust, conditional plans).

Apart from these crucial application requirements, we have also to take into account the following ones, referred to the way in wich an expert works:

- Expressiveness: the ontology used to represent planning domains must be expressive enough to neatly represent evey crucial aspect in this domain. In addition, some kind of assisting tool should be provided in order to facilitate expert tasks like domain description or posing high-level problems.

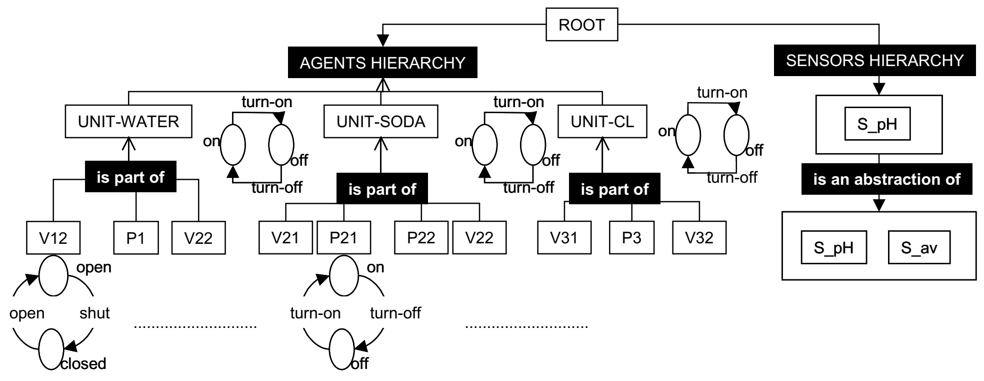

Figure 3: A compositional hierarchy of agents, and a sensor hierarchy representing a knowledge-based description of the layout of the manufacturing system shown in Figure 1

- Mixed autonomy: the expert may require the system to be capable of autonomously making as many complex decisions as possible. However, in some cases the expert might require to have a complete control over the system. In any case, it is necessary to find a balance between complete autonomy and an agile and interactive process, facilitating and reducing the human effort in both, domain description and problem solving phases.
- Suitability of the results: the plans obtained should be directly (easily) translated into known control-specification formalisms (Petri Nets (Peterson 1981) or Graphcets (IEC 1988)). This would mean that either the final plans or the intermediate plans (considering a mixed-autonomy planning process) should also be understandable by human experts, what should facilitate both, the justification of the decisions made, and the human interaction with the system.

It is well known that all of these requirements are very difficult to satisfy if one focus only in one single planning technique to cope with the real domains here addressed. On the one hand, it is well known that the use of pure HTN (Yang 1997) techniques in manufacturing applications (Nau, Gupta, & Regli 1995; S.Viswanathan *et al.* 1998), leads to the development of planning approaches in which the management of a great amount of procedural knowledge reduces the deliberative capabilities of the planner and, in addition, increases the expert effort in the description of domains (Biundo & Schattenberg 2001; Castillo, Fdez-Olivares, & González 2001; Kambhampati, Mali, & Srivastava 1998). This also results in a weak concept of autonomy: plan construction can only rely on the procedures introduced, so general problem solving capabilities are strongly limited.

On the other hand, POCL-based (Weld 1994) approaches supply general problem solving capabilities, what is useful to increase the degree of autonomy, and they offer a sound semantical basis for obtaining correct plans, that can be translated into suitable representation formalism of control software (Castillo, Fdez-Olivares, & González 2000a; 2000b). However, they lack of the necessary hierarchical framework to follow present industrial standards, therefore, a POCL planning process is not so suitable as one could wish.

Finally, it seems that conditional (or contingent)(Onder & Pollak 1999; Pryor & Collins 1996) planning is a very appropriate technique for a realistic design of robust control software (Bresina *et al.* 2002; Castillo, Fdez-Olivares, & González 2002). However, these techniques are scarcely applied in automated manufacturing operation. Concretely, we are not aware of any application of these techniques to automated manufacturing. The reason is that conditional approaches rely on a very simple model of non-durative actions in which concurrent actions are not allowed, and they do not scale up to large problems, what affects negatively to the agility and reliability of a planning application in this real domain.

Next we will show how most of these requirements can be met by means of the integration of a hybrid hierarchical planning approach introduced in (Castillo, Fdez-Olivares, & González 2001), and a conditional planner presented in (Castillo, Fdez-Olivares, & González 2002). Then, we will show how to integrate this system into a software desing environment in order to solve software design problems following standard manufacturing procedures.

Integrating hierarchical and conditional planning

According to the concepts previously explained, a planning domain is described as a *compositional hierarchy of agents*, where agents are described at different levels of abstraction, and their behaviour is described as a finite state automaton. Agents at the lowest level of the hierarchy are called primitive agents, and higher level agents are called compound agents, in such a way that a relationship of type *is-part-of* is defined between them.

Another key entities in the domain ontology are sensors, used to represent those sources of uncertainty of the real-world that can be detected by sensory equipment and such that they affect the behaviour of agents and their influence may be somehow foreseeable. Sensors are also hierarchically distributed in a *sensor hierarchy*, different from the agent hierarchy, where every level of abstraction i contains a set of sensors that affect the agents of level i. Such a hierarchy is built by an expert, at the domain description stage, following a simple rule: given a compositional hierarchy of n levels of abstraction, the whole set of sensors must appear in the lowest level of the sensor hierarchy (the level n), and for the remaining levels, every level i may contain a smaller set of sensors than the level j, $i < j$, by simply dropping sensors from level j.

The compositional hierarchy of agents and the sensor hierarchy for the example explained above are shown in Figure 3. In this figure, we can see that the expert has decided that the behaviour of agents of the highest level (whose behaviour is much simpler than that of their components) can only take into account the existence of the sensor of pH (avoiding the influence of the sensor of availability of pumps) what should simplify the design process for a high-level control procedure for the neutralization of water.

Thus, starting from such as domain description, the general idea underlying the planning process is very well known and assimilable by any expert: the design and development of a hierarchical and closed-loop control program corresponds to the construction of a hierarchical and conditional plan whose different levels of abstraction are conditional plans, and whose actions are represented at different levels of granularity. Thus, every conditional plan at a level of abstraction i, is built on the basis of a POCL and conditional process. Its abstract actions can be seen as high-level modules, containing conditional structures between them, used for making level-i decisions, according to the sensors used at level i. Furthermore, every abstract action α at level i will be later decomposed into a set of causal actions with conditional structures, at level $i + 1$, inside the scope of that high-level module α. As a consequence, the final result is a very close representation of a modular, conditional control program, built under a top-down process following current industrial standards.

Next, we will describe in detail the most relevant aspects of the planning entities, as well as the conditional and hierarchical process.

Basic concepts on the planning entities

Every sensor is associated with a discrete set of possible states. For example, the sensor of pH is represented as a sensor, SpH, which can take one of the following states $\{Acidic, Neutral, Basic\}$. The sensor of availability Sav, can take one of the states $\{1, 2\}$ and informs about the availability of either pump P21 or pump P22.

As described above, the problem consists on obtaining neutral water in the tank T2. The recipe for this high-level process is represented by the two high-level goals (PH *neutral*) and (CONTAINS *water T2*). In addition, it is important to take into account that the pH is initially unknown, but suitable to be detected by the sensor SpH. This is represented in the initial state, at the level of abstraction 1, by means of the literal (PH *!ph*), where *!ph* stands for a *run-time variable* used to represent the incomplete knowledge about the pH of the water. A run-time variable is always associated with a sensor, in such a way that every possible value of the variable is associated with a single possible state of its sensor. Thus, in this case, the domain of *!ph* is $\{Acidic, Neutral, Basic\}$. In addition, since sensors are hierarchically distributed and SpH is the only sensor described at level 1, the sensor of availability do not affect to the planning process at this level of abstraction.

The behaviour of every agent is represented by means of a finite state automaton where states represent internal states of the agent and transitions represent causal actions that the agent might execute. Every action can be either *primitive* or *compound* (depending on which agent it is attached to, compound or primitive), and they share the same structure at every level of abstraction: a set of previous, simultaneous, and query requirements (Castillo, Fdez-Olivares, & González 2000a) which represent different types of conditions that must hold in order to execute the action, and a set of effects describing changes in the environment and in the internal state of the agent.

Furthermore, every primitive or compound action is durative (Fox & Long 2002), in such a way that the duration of an action a is represented as an *interval* of actions, $[a, \mathit{End}(a)]$, defined from a until another action $b = \mathit{End}(a)$. Both actions belong to the same agent g and they are such that they produce a consecutive change of state in g. Simultaneous requirements are then used to represent conditions which must hold *during* an interval of actions (like invariant conditions in PDDL 2.1).

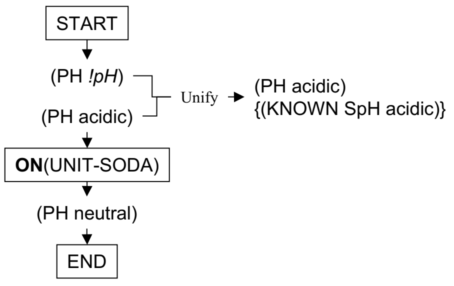

Figure 4: Unification of literals with run-time variables

Causal actions can also produce different effects depending on states of sensors. For example, the dummy action **START** is a causal action that contains a literal with a run-time variable in its effects which represents the partially known initial state (see Figure 4). This introduces the problem of how to represent at planning time both, the set of possible outcomes of a causal action, and the conditional courses of action induced by the states of sensors.

Decide actions are used to cope with these issues. A decide action is also associated with a sensor, and it is automatically generated at planning time when a non-deterministic causal action (which may be either the action **START** or any other action of an agent) is used to satisfy an open condition. For example, let us consider the top of the hierarchy shown in Figure 3 and suppose that the action **ON**(UNIT-SODA) has been inserted to satisfy the highest level goal (PH *neutral*). Then, its open precondition (see Figure 4) may be covered by the non-deterministic action **START**. However, depending on the state of the sensor SpH this action will result in one of the following possible outcomes: (PH *acidic*),(PH *neutral*), or (PH *basic*). Therefore a decide action associated to the sensor SpH is generated and attached to the action **START**, representing both, its possible outcomes, and the three possible courses of action depending on the state of the sensor of pH (See Figure 5).

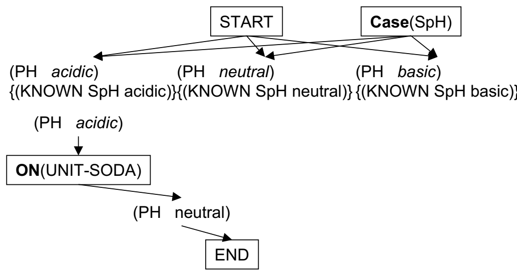

Figure 5: A decide action and the possible outcomes of action **START**

Another key issue is concerned with the unification of literals containing run-time variables. In the previous example, the planning process detects that the action **START** is a producer for the open condition of the action **ON**(UNIT-SODA) by means of the unification of the literals (PH *!ph*) and (PH *acidic*) (See Figure 4). This unification results in a literal completely instantiated, (PH *acidic*) but, indeed this literal will represent an executability condition of the action **ON**(UNIT-SODA) only when the sensor SpH is in the state $acidic$. That is, the literal will be true and, therefore, the execution of the high level action **ON**(UNIT-SODA) will be possible, only when the sensor SpH is known to be in the state $acidic$. This means that sensors restrict the executability of actions, what leads us to introduce the concept of *knowledge restrictions* in order to represent this sensor-dependency.

A knowledge restriction is represented as a special literal $(\text{KNOWN}\ \sigma\ u)$, where σ stands for a sensor and u for one of its possible states, and they become an important part of an annotated representation of literals: a literal is represented as $(atom\ .\ kr)$ where $atom$ stands for the atom of l and kr stands for a *set of knowledge restrictions*. For example (see Figure 4), the resulting literal of the previous unification is

$$((\text{PH } \textit{acidic}).\{(\text{KNOWN}\ \ SpH\ acidic)\})$$

and it is interpreted as the literal is true (the pH is acidic), when the sensor SpH is in the state $acidic$[1]. It is important to note that knowledge restrictions are not described by the expert, but generated at planning time whenever a run-time variable unifies with a constant. Therefore the concept of knowledge restrictions introduces new modes of satisfaction, the fundamental mechanism of the one-level conditional planning algorithm that we will illustrate next.

[1]An empty set of knowledge restrictions, of a literal or an action, means that no sensor restricts them.

The one-level conditional planning process

Every level of abstraction i in the compositional and sensor hierarchy represents a piece of the domain that is given as input to a conditional planning algorithm (called ADVICE (Castillo, Fdez-Olivares, & González 2002)) that has to build a level-i correct conditional plan . That is, a plan containing conditional structures and whose causal actions are all executable in some context. A causal action is executable when all its preconditions are satisfied by the effects of another causal action. As seen above, the satisfaction of the literals that represent preconditions is based on an unification algorithm that might produce new knowledge restrictions. As a consequence, the concept of knowledge restrictions introduces new modes of satisfaction that are worth to note.

It can be seen in Figure 5 that the set of knowledge restrictions of an instantiated effect of the non-deterministic action START subsumes the set of knowledge restrictions of an instantiated precondition of **ON**(UNIT-SODA), this is a mode of literal satisfaction called *possible satisfaction*. As explained above, the execution of **ON**(UNIT-SODA) is only possible in the *conditional context* in which the sensor SpH is known to be in the state $acidic$. Hence, in order to describe that this action can only be executed in such as conditional context, the set of knowledge restrictions $\{(\text{KNOWN}\ \ SpH\ acidic)\}$ is propagated forwards, towards the action **ON**(UNIT-SODA) and its effects (actions are also annotated with knowledge restrictions to represent the conditional context where they have to be executed). This propagation is recursively performed throughout the causal structure of a conditional plan (See Figure 6).

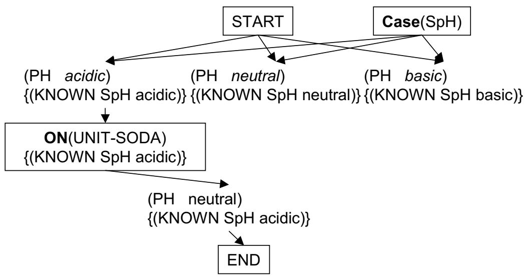

Figure 6: Forward propagation of knowledge restrictions.

However, there are some situations in which the forward propagation is not further possible (for example, in the case of the goal literal (PH *neutral*)). Thus, in these cases we interpret that the literal cannot be completely satisfied in the single context represented by the knowledge restrictions being propagated ($\{(\text{KNOWN}\ \ SpH\ acidic)\}$) and that it should also be satisfied in all of the complementary contexts with respect to this set, that is, $\{(\text{KNOWN}\ \ SpH\ basic)\}$ and $\{(\text{KNOWN}\ \ SpH\ neutral)\}$. As a way to support this *necessary satisfaction* of a literal, we use the concept of *conditional expansion* of a literal: a set of annotated literals representing all the disjoint contexts (branches) which have to be considered to achieve a same goal (Figure 7 shows the conditional expansion of the literal (PH *neutral*), referred to the forward propagated knowledge restrictions).

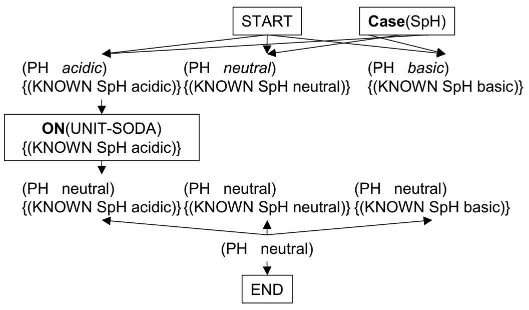
START
Case(SpH)
(PH acidic) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH neutral)}
(PH basic) {(KNOWN SpH basic)}
ON(UNIT-SODA) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH neutral)}
(PH neutral) {(KNOWN SpH basic)}
(PH neutral)
END

Figure 7: Conditional expansion of a literal

This conditional expansion has also a very important role: the set of literals so obtained represents a set of open conditions that have to be satisfied in every different conditional context, therefore, every literal generates a conditional branch, inside a conditional plan, that will be built by means of a POCL regressive process.

Figure 8 shows both concepts: how these conditional branches are generated by the *conditional expansion* of the goal (PH *neutral*), and how every expanded literal is satisfied in every different branch. As can be seen in this Figure, the set of knowledge restrictions of an instantiated effect of the causal action **ON**(UNIT-CL) is already included in the set of knowledge restrictions of the conditionally expanded literal (PH *neutral*). This is a mode of literal satisfaction called *circumscribed satisfaction*. This means that the literal will only be true when the action **ON**(UNIT-CL) is executed under the conditional context represented by the set of knowledge restrictions $\{(\text{KNOWN} \ \ SpH \ basic)\}$.

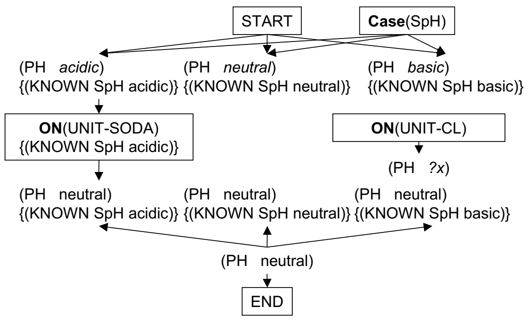
START
Case(SpH)
(PH acidic) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH neutral)}
(PH basic) {(KNOWN SpH basic)}
ON(UNIT-SODA) {(KNOWN SpH acidic)}
ON(UNIT-CL)
(PH ?x)
(PH neutral) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH neutral)}
(PH neutral) {(KNOWN SpH basic)}
(PH neutral)
END

Figure 8: Cirumscribed satisfaction.

Because of this, the knowledge restrictions of the unification must be propagated backwards, back to the consumer action (See Figure 9). When the knowledge restrictions are propagated, the open precondition of the action **ON**(UNIT-CL) is also satisfied in a circumscribed mode by the action **START** (throughout the decide action associated to the sensor SpH). Finally, when all of the expanded literals of the conditional expansion of the goal (PH *neutral*) are completely satisfied in a circumscribed mode, it is said that this literal is *necessarily satisfied*.

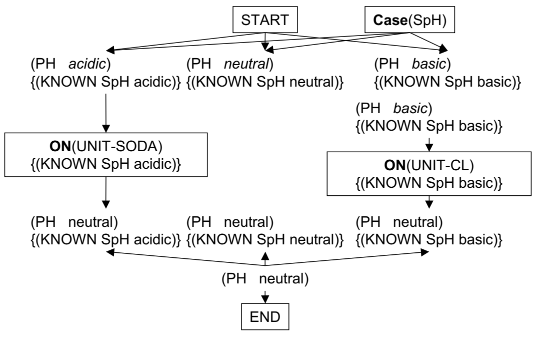
START
Case(SpH)
(PH acidic) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH neutral)}
(PH basic) {(KNOWN SpH basic)}
(PH basic) {(KNOWN SpH basic)}
ON(UNIT-SODA) {(KNOWN SpH acidic)}
ON(UNIT-CL) {(KNOWN SpH basic)}
(PH neutral) {(KNOWN SpH acidic)}
(PH neutral) {(KNOWN SpH neutral)}
(PH neutral) {(KNOWN SpH basic)}
(PH neutral)
END

Figure 9: Backward propagation of knowledge restrictions.

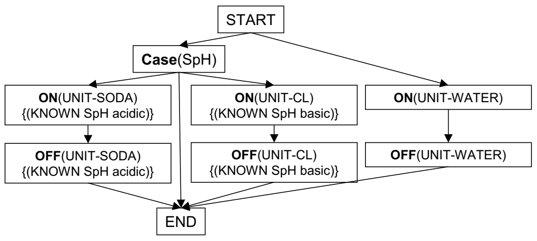
START
Case(SpH)
ON(UNIT-SODA) {(KNOWN SpH acidic)}
ON(UNIT-CL) {(KNOWN SpH basic)}
ON(UNIT-WATER)
OFF(UNIT-SODA) {(KNOWN SpH acidic)}
OFF(UNIT-CL) {(KNOWN SpH basic)}
OFF(UNIT-WATER)
END

Figure 10: The final conditional plan at level 1.

Thus, the final conditional plan so obtained is shown in Figure 10. This plan contains several compound causal actions, that can be seen as high-level modules of a more detailed hierarchical program, and a single conditional structure, made up of the decide action **CASE**(SpH), that is used to represent that, at running time, a conditional decision on the known state of the sensor SpH has to be made: in the case of this state to be $acidic$, the unit process UNIT-SODA will be turned on in order to neutralize the water in the tank T2, in the case of the state to be $basic$ the unit process UNIT-CL will be turned on, and nothing has to be done in the case that the pH is neutral.

In addition, in order to represent the causal structure of a conditional plan, at a single level of abstraction, the following types of causal links are used:

- *Possible link*: they are causal links used to protect a literal that belongs to a conditional expansion. These causal dependencies are very important in the conditional planning process because the literals so protected represent a stop condition for the forward propagation of knowledge restrictions. Thus, not only the final action of a plan is a stop condition for the recursive forward propagation of knowledge restrictions, but also any action supported by a possible link, what provides the support for creating conditional nested branches.
- *Normal link*: they are used to protect any other literal in the plan.

Finally, the conditional planning algorithm is based on a planning by refinement paradigm where the following refinement tasks may arise:

- *Threats*, that are solved by promotion or demotion on the different types of causal dependencies.

- *Open conditions*, that are solved depending on the mode in which an action satisfies an open condition, that is
  1. in the case of a circumscribed mode, a backward propagation of knowledge restrictions is performed
  2. in the case of a possible mode, a forward propagation is performed recursively throughout the causal structure. But, it is important to take into account that those literals in which recursive propagation is not possible are recorded into a set of fails.

  In addition, if an open condition is solved by a non-deterministic action with an unknown effect (containing a runtime variable) a decide action is generated (or reused) to represent in the conditional plan the possible outcomes of that non-deterministic action.
- *Conditional branching*: when the forward propagation of knowledge restrictions associated to a literal fails, it will be used to build new conditional branches by means of its conditional expansion, so every newly expanded literal becomes an open condition for every new branch.

This conditional planning algorithm presents some features that make it suitable for its application to the domain of interest in this work: it is based on an very expressive model of actions, where actions are durative; non-deterministic actions are dynamically managed at planning time and there is no need to know all their possible outcomes at domain description or planning time; finally, conditional plans obtained are represented as a DAG and not as a tree, like most of conditional approaches (Weld, Anderson, & Smith 1998). These features, together with the conditional process, will allow to meet requirements as robustness, reliability and suitability of the results.

Once a conditional plan at the highest level of abstraction (level 1) has been built, a stepwise hierarchical refinement is performed on every action at this level, in order to build a more detailed conditional plan (at level 2). As we will see in next section, the integration of this algorithm into a hybrid hierarchical process will lead to a conditional planning process that also meets requirements like expressiveness, agility, and scalability to large problems.

Decomposition of high-level modules

Every compound action α at a level of abstraction i has attached a set of *expansion methods*, where every expansion method is a set of literals, at level $i + 1$, representing subgoals to be achieved by actions at level $i + 1$. The set of literals of a given expansion method m_j, of a compound action α attached to a compound agent g, are obtained at planning time by means of an *articulation function*(Castillo, Fdez-Olivares, & González 2001) that translates the level-i effects of α into a set of level-($i + 1$) literals. The application of this articulation function, for every compound agent g, is supported by the *interface* of g: a set of association rules included by an expert at the domain description stage to describe how the level-i literals of every compound action, attached to g, are related with level-($i + 1$) literals.

For example, the interface of the agent UNIT-SODA may contain the following association rule:

(PH *neutral*) → {(FLOW *soda TSoda T2*), (CONTAINS *soda T2*)}

meaning that the high-level effect "UNIT-SODA has neutralized the pH" of action **ON**(UNIT-SODA) is translated into a more detailed goal (at a greater granularity) "obtain a flow of soda between TSODA and T2, and supply soda to tank T2", which has to be achieved by actions of agents at the level of abstraction 2.

Furthermore, since actions and literals at a given level of abstraction i may be annotated with knowledge restrictions coming from upper levels, actions and literals of level $i + 1$ may inherit these knowledge restrictions thanks to the articulation function, by means of a simple process. For example, at level 1, action **ON**(UNIT-SODA) and its effects are annotated with the set of knowledge restrictions $\{(\text{KNOWN}\ \ SpH\ acidic)\}$, hence the result of the translation of this literal is the set of literals

$$\{(\text{FLOW}\ \textit{soda TSoda T2})\ .\{(\text{KNOWN}\ \ SpH\ acidic)\},$$
$$(\text{CONTAINS}\ \textit{soda T2}).\ \{(\text{KNOWN}\ \ SpH\ acidic)\}\}$$

by simply copying the set of knowledge restrictions from level 1 to level 2.

This mechanism for both, the translation of literals and the inheritance of knowledge restrictions, across different levels of abstraction, allows every compound action α, at level i, to be *dynamically decomposed* into a set of subactions at level $i + 1$, following two steps:

1. Determine the set of annotated literals at the next level $i + 1$ which have to be achieved by actions of agents at that level.
2. Determine, by means of POCL-based techniques, the set of actions such that either they satisfy those literals generated by α or they contribute to their establishment. These actions will make up the real decomposition of α.

While the first step is a simple translation mechanism (mainly supported by the articulation function), the second one is a key step for the construction of a hierarchical plan (that finally will be used as a hierarchical control program), and for understanding the advantage of using a hybrid approach. At this step a modularity relationship between two plans $\mathcal{S}^i$ and $\mathcal{S}^{i+1}$ at different abstraction levels is established, so that every action in $\mathcal{S}^i$ is mapped into a disjoint set of actions of $\mathcal{S}^{i+1}$. The hybrid hierarchical process uses the function *Scope*(a), to represent the "is composed by" hierarchical relationships between actions. The definition of this function is based on a regressive process of literal satisfaction wich includes the propagation processes of knowledge restrictions above described. Thus, it is possible to dynamically generate, at planning time by means of POCL and conditional techniques, the set of subactions for a given compound action at level of abstraction i. Summarizing, the scope of an action a, namely *Scope*(a), is a higher level action α if one of the following conditions holds:

1. a establishes one or more literals generated by different higher level actions, $\{\alpha_1, \ldots, \alpha_n\}$, and α is one of the first actions of this set (See Figure 11).
2. a establishes several requirements of actions at its same level, $\{a_1, \ldots, a_n\}$, and α is the scope of one of the first actions of this set (See Figure 12).

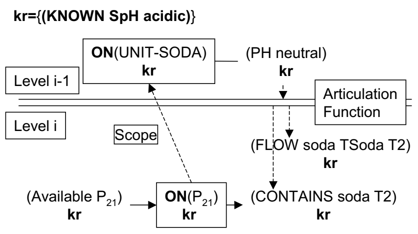

Figure 11: Dynamically deciding the scope of an action (Condition 1).

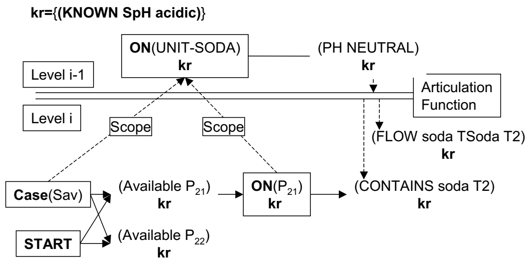

Figure 12: Dynamically deciding the scope of an action (Condition 2).

Figures 11 and 12 also show how knowledge restrictions can be inherited from a level i to a level $i+1$, and how they can be propagated throughout the actions of level $i+1$. That is, when the level-2 action **ON**(P21) satisfies in a circumscribed mode the translated literal, the backward propagation of knowledge restrictions ensures that all the actions inside the scope of the high-level module **ON**(UNIT-SODA) will be executed in its same conditional context. On the other hand, due to the existence of new sensors at the level 2, the incomplete initial state a this level is increased, what allows for the introduction of new decide actions inside the scope of the action **ON**(UNIT-SODA), resulting in a conditional plan with nested and well organized conditional structures.

This decomposition process preserves the expressiveness of decomposition rules of other models, that is, it allows for alternative action decompositions, but it improves them because it allows a compound action to be related with submodules and conditional structures of different grain size. Additionally, it does not require to know the modular decomposition of every compound action prior to the planning process (during domain description stages), and in most cases the only required knowledge is the interface of every compound agent, covering much better requirements of simplicity and autonomy.

It must also be said that "abstract" decide-actions used to represent high-level conditional structures, at a given level i, are directly copied into the next level of abstraction $i+1$. Thus, the conditional structure of a conditional plan at level i is completely inherited by its next level of abstraction.

Another feature of this mechanism is that the correctness of a decomposition does not have to be checked "a priori" by hand during domain description. Instead of this, this task is shifted into the planning process who becomes the responsible of dynamically checking the correctness of every decomposition at every level of abstraction.

Guaranteeing the correctness of a hierarchical plan

The dynamical decomposition introduces the problem of how to determine, at planning time, that a decomposition is valid. That is, not every syntactically valid modularization is also a semantically valid one, so a set of criteria which feature semantically valid modularity relationships should be defined taking into account the following issues:

- Every action at level $i+1$ must be related to an action at level i.
- A decomposition of an action should not have any internal irresoluble conflict.
- Subactions should not interfere with the higher-level effects of the action whose decomposition they belong to.

In HTN approaches these issues are responsibility of the human who writes a domain description, decreasing the autonomy of the planner, and assigning the most important task of domain validation to the control engineer. This shift in responsibility could be avoided by defining general semantic properties which must be satisfied by any modular decomposition, and giving more responsibility to the planner to check these semantic properties and, therefore, to find a valid decomposition. Then, our planner, named ADVICE-H , decides whether a decomposition is valid just by looking for harmful causal interactions between actions at the same level of abstraction (by means of classical POCL techniques (Castillo, Fdez-Olivares, & González 2001)), and also between actions at different abstraction levels.

With respect to causal dependencies between actions at different levels of abstraction, the concept of *hybrid causal link* is used (Castillo, Fdez-Olivares, & González 2001). A hybrid causal link $[a_\alpha \xrightarrow{l} \beta, \gamma]$ is a structure used to represent that a literal expanded by α and satisfied by a_α, a subaction of α, has to be protected from any other subaction of α, in such a way that none of these subactions could negate that literal. Hybrid causal links are also used to detect and solve *hybrid threats*, which take into account the existence of interactions between actions at different levels of abstraction. Hence, the causal consistency across different levels of abstraction, and the correctness and reliability of the hierarchical process is guaranteed.

Finally, following the planning process described, a hierarchical, modular, and conditional plan is obtained as the one shown in Figure 13. As can be seen, the high-level modules at level 1 has been decomposed into lower-level actions and they also contain encapsulated conditional structures.

Integration into a software development environment

Apart from the above described hybrid hierarchical and conditional planning process, our system incorporates some functionalities which help to facilitate its integration into a real integrated development environment for control software.

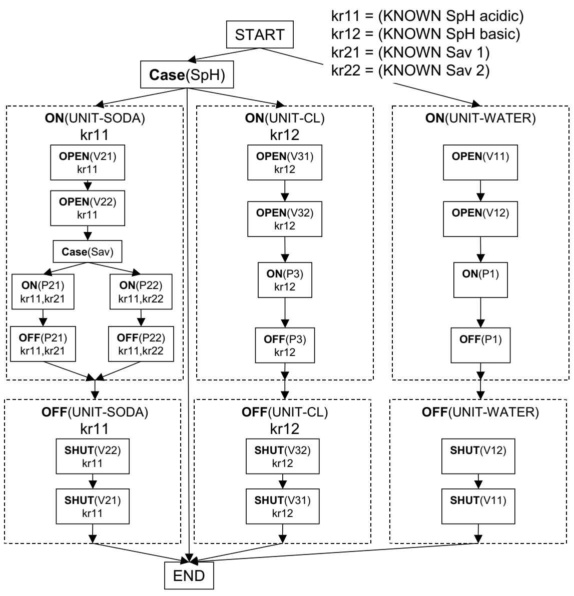

Figure 13: The final hierarchical plan for the problem formerly introduced.

First, from the point of view of the expert at the input stage, the elaboration of a SP88-specification of a plant layout is simplified by means of a *Library of agents* where generic agents with generic behaviour are defined. Thus, describing a new agent as part of a given plan layout can be seen as a *drag and drop* process in which the expert takes a generic agent from the Library and then he/she instantiates it with specific knowledge for that concrete plant. This kind of assisting tool simplifies domain description tasks and should allow for a reduction of the time expent in this stage.

Second, at the planning-process stage, ADVICE-H incorporates a visual interface that allows an expert to interact during both, the knowledge description phase and the problem-solving phase, in such a way that the expert can guide the problem solving phase. Thus, due to the hierarchical planning process, and the hierarchical distribution of sensors at different levels of abstraction, the expert can help ADVICE-H either to make complex conditional decisions at higher levels of abstraction, or to organize the nested conditional structures. On the other hand, ADVICE-H can help the expert in lower-level complex reasoning processes, showing what whould happen when an explicit decision is taken ("what-if" exploration). For example, ADVICE-H might decide the right set of valves to be open given that a high-level task of transportation has been determined, thanks to its capability to dynamically obtain the decomposition of a high-level task.

Finally, with respect to the output stage, ADVICE-H is based on a POCL planning process capable of translating every plan obtained at every level of abstraction into known standard formalisms of control software, as Graphcet (Castillo, Fdez-Olivares, & González 2000a) of Petri Nets (Castillo, Fdez-Olivares, & González 2000b). Thus the suitability of the hierarchical conditional plans obtained offers a double advantage: the expert not only will be able to understand much better the results presented at different levels of abstraction, but also the plans can be translated level by level into known formalisms for the specification of control programs, what allows ADVICE-H to bridge the gap between the planning techniques used and a real software development environment.

Advantages from the application perspective

Problem	Domain size	Sensors	Plan lenght	CPU time
1	12 agents	2	25	400sg
2	27 agents	5	53	1100sg
3	11 agents	2	34	200sg
4	28 agents	1	61	157sg

Table 1: Experimental results of ADVICE-H in several control software design problems for different manufacturing systems.

Table 1 shows some results about the performance of ADVICE-H seen as an autonomous planning system to solve control software design problems. *Problem 1* is the problem ilustrated in this paper and *Problem 2* is an extension of this manufacturing system with the same high-level process but more complex subprocesses. *Problem 3* consists on a transportation problem with a single resource to be shared, and where the content of the source tanks is unknown. *Problem 4* is a more complex manufacturing system in which the process carried out depends on the product to be obtained.

These results show that ADVICE-H is not as efficient as the planners that take part in the planning competition. However ADVICE-H is not intended to work autonomously, but following an interactive process with the expert in order to solve a control software design problem. In this case, the time expent by ADVICE-H is not significant compared with the time an expert takes to design a correct and robust control program (it may take hours, even days). In this framework, ADVICE-H can be used either as an off-line rapid prototyping assisting tool for a new given design problem, or as an off-line rapid reconfiguration tool, in case of any modification of the plan layout.

Thus, the main advantage of using ADVICE-H comes from its usage as a tool that saves engineering time, following the same standard processes than those followed by manufacturing experts. This is specially important if we take into account that the task of designing and developing a control program represents a 35 percent of the whole control operation costs in a manufacturing system (the 40 percent corresponds to hardware costs and the 25 percent to comissioning).

Therefore, we argue that the capability of obtaining sound results, shown in an understandable manner, following a reliable, mixed-autonomous hierarchical/conditional process, on an expressive knowledge representation, are AI Planning technological developments that reduce that great percentage of engineering time, and that will be welcome in manufacturing business, although the time performance of the techniques here presented is not optimal with respect to the average of planners in the planning competition.

Conclusions and lessons learned

In this work we have presented a completely novel AI planning application that integrates POCL, hierarchical and conditional planning techniques, in order to fit a set of analyzed requirements which have to be met by the design of control programs in the field of Automated Manufacturing. Although the integration of these techniques may overcome some shortcomings detected in the application of previously developed AI Planning technology to this field, we have shown that this is not sufficient if one take into account the way in wich experts solve design problems. Therefore, the core of the system has had to be extended with other capabilities in order to fit present manufacturing standards: a tool to facilitate domain descriptions, an interface that supports an interactive problem solving process, and the necessary translation mechanisms to obtain understandable plans. All of these techniques are intended to reduce the negative impact, produced on the experts, due to the use of new technologies for solving known complex problems.

During the development of this application some lessons have been learned: first, it is necessary to have a deep understanding of the domain application in order to face the challenge to develop real-world AI Planning techniques. This involves to know the standard procedures followed by experts in order to perform reliable planning processes from their point of view. In addition, we have realized that not always standard planning languages are ready-to-use tools for real-world. For example, neither the use of sensors, nor the compositional hierarchy of agents, nor the dynamical decomposition of abstract actions are addressed in current standard planning laguages, however, they are key issues that facilitate many expert tasks in the design of control programs for manufacturing systems.

References

Aylett, R.; Petley, G.; Chung, P.; Soutter, J.; and Rushton, A. 1997. Planning and chemical plan operation procedure synthesis: a case study. In *Fourth european conference on planning*, 41–53.

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief. a preliminary report on flexible integration on nonlinear and hierarchical planning. In *Proceedings of 6th European Conference on Planning (ECP-01)*.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous and resource uncertainty: A challenge for AI. In *Proceedings of AIPS02. Workshop on temporal domains.*, 91–97.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2000a. Intelligent planning of grafcet charts. *Journal of Robotics and Computer Integrated Manufacturing systems* 16:225–239.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2000b. A three-level knowledge-based system for the generation of live and safe petri nets for manufacturing systems. *Journal of Intelligent Manufacturing* 11(6):559–572.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2001. On the Adequacy of Hierarchical Planning characteristics for Real-World Problem Solving. In *Proceedings of the Sixth European Conference on Planning ECP-01.*

Castillo, L.; Fdez-Olivares, J.; and González, A. 2002. A conditional approach for the autonomous design of reactive and robust control programs. In *Sixth International Conference on AI Planning and Scheduling AIPS02. Proceedings of the Whorkshop on Real-World Planning*.

Fox, M., and Long, D. 2002. PDDL2.1: An extension to PDDL for modelling time and metric resources. In *Proceedings of AIPS-02.*

IEC. 1988. Preparation of function charts for control systems. Technical Report IEC-60848, International Electrotechnical Commission.

ISA. 1995. *Batch control Part 1: models and terminology (SP-88)*. Instrument Society of America (ISA).

Kambhampati, S.; Mali, A. D.; and Srivastava, B. 1998. Hybrid planning for partially hierarchical domains. In *Proceedings of AAAI-98/IAAI*, 882–888.

Klein, I.; Jonsson, P.; and Backstrom, C. 1998. Efficient planning for a miniature assembly line. *Artificial Intelligence in Engineering* 13(1):69–81.

Nau, D.; Gupta, S. K.; and Regli, W. C. 1995. AI planning versus manufacturing-operation planning: A case study. In *IJCAI-95*, 1670–1676.

Onder, N., and Pollak, M. E. 1999. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proceedings of AAAI-99*.

Peterson, J. L. 1981. *Petri nets theory and the modelling of systems*. Prentice-Hall.

PLANET. 1998. European Network of Excellence in Artificial Intelligence Planning, technical coordination unit in intelligent manufacturing. `http://odl.education.salford.ac.uk/dan/pims/home.html`.

PLANET. 2001. The PLANET roadmap on AI planning and scheduling. http://www.planet-noe.org.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision based approach. *Journal of Artificial Intelligence Research* 4:287–339.

S.Viswanathan; C.Johnsson; R.Srinvivasan; V.Venkatasubramanian; and Arzen, K. 1998. Automating operating procedure synthesis for batch processes. part I: Knowledge representation and planning framework. *Computers and Chemical Engineering* 22:1673–1685.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of AAAI-98*.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4).

Yang, Q. 1997. *Intelligent Planning. A decomposition and Abstraction Based Approach*. Springer Verlag.