A hybrid hierarchical/operator-based planning approach for the design of control programs

L. Castillo and **J. Fdez-Olivares** and **A. González** [1]

Abstract.

The design of a control program is a complex process whose result must satisfy very restrictive constraints imposed by new manufacturing systems needs as flexibility, quick response, correctness and low-cost building process. Current AI Planning approaches for the synthesis of control programs are proving to be very useful to satisfy these needs. But they have to be extended in order to build efficient and realistic systems which obtain truly real world solutions. This work presents an approach in this direction which mixes hierarchical and POCL techniques in order to build an architecture closer to the way that control engineers reason in order to design a control program. The utility of this approach is shown along this paper.

1 INTRODUCTION

The design of a correct and complete industrial control program is a process which involves different sources of knowledge and whose final result is a sequence of control actions [6]. Concurrency, conditional branches, soundness, security and flexibility are some of the features that these sequences are expected to have.

The design process of a control program with these features is very complex, even for human programmers. Traditionally control engineers use different methodologies, standards, formal tools and computer utilities to carry out this task. The ISA-SP88 [13] standard (Figure 1) is one of such methodologies used to hierarchically design control programs for manufacturing systems. This standard allows for a hierarchical specification of physical, process and control models of a manufacturing system.

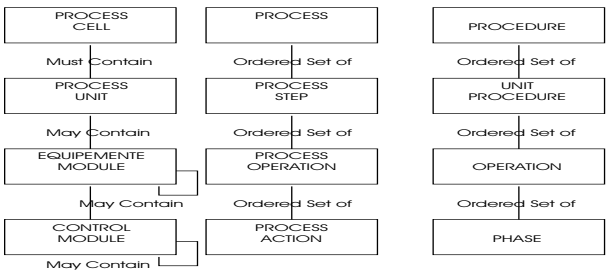

Figure 1. Physical, Process and Control Model of SP88.

[1] Departamento de Ciencias de la Computación e Inteligencia Artificial, E.T.S. Ingeniería Informática. Universidad de Granada, 18071 Granada, SPAIN, email: L.Castillo,Faro,A.Gonzalez@decsai.ugr.es

Starting from a hierarchical physical model and from a process specification (recipe) at the higher abstraction level, a control engineer obtains a modular control program at different levels of granularity. There are formal tools as GRAFCET [12] for the representation and specification of such programs.

These methodologies used by control engineers to develop control programs are useful and necessary, buy they are not sufficient if we take into account requirements as flexibility and quick response in new generation manufacturing systems [7]. The application of AI Planning techniques for the synthesis of control programs or operating procedures in manufacturing systems is a promising technology that meets these requirements. Although at present it is in an initial stage [5], interesting approaches have been carried out ([1, 6, 14, 20]). These techniques are proving to be very useful, allowing for an error-free, fast and low-cost building process of control programs.

Not all of these approaches are based on hierarchical planning techniques [8, 10, 18], which are useful to represent the hierarchical structure of devices and their operation in manufacturing systems. In addition, hierarchical techniques are closer to the way in that control engineers

- represent a control program (modular and hierarchically), and
- reason in order to find the sequence of instructions of the control program.

In this sense, this work presents a planning approach which employs hybrid POCL and hierarchical planning techniques in order to

- Represent an industrial plant as a device hierarchy at different levels of granularity, which accepts SP88 descriptions, providing a friendly input level for control engineers, and
- autonomously develop control programs for manufacturing systems following a hybrid planning process (POCL+hierarchical), which results in a hierarchy of control sequences (plans) at different levels of detail, closer to the way that humans develop modular industrial control programs and, thus, providing a more understandable output.

In the next section we will show some related work and, afterwards, we will describe our approach.

2 RELATED WORK

Apart from the partial-order planning approaches mentioned in previous section, one of the more recent hierarchical approaches can be found in [20]. It is a general planning framework for the synthesis of operating procedures following a top-down methodology. The knowledge representation scheme is a translation of the SP88

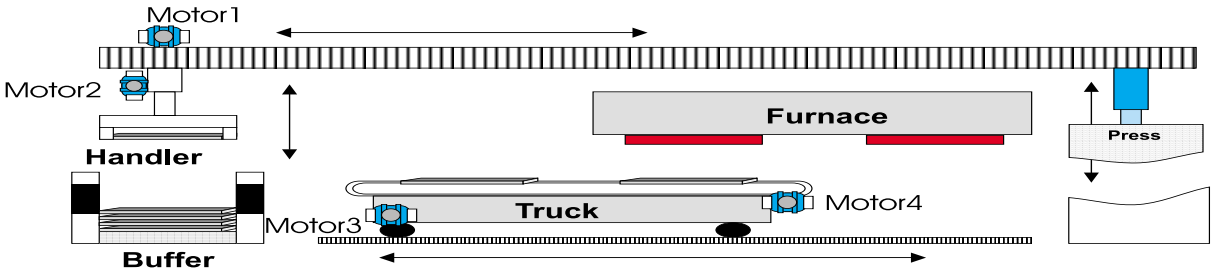

Figure 2. A Manufacturing System.

standard, which allows for creating a *procedural knowledge base* at different abstraction levels. The planner then applies basic HTN [8] techniques in order to find a low level procedure which meets the process of products introduced as problem.

In this system the user has to introduce a great deal of knowledge to solve a problem, and the main role is left to the representation and management of the procedural knowledge base. Therefore, the planner, and thus the autonomy of the approach, is only a very small part of the whole system, and its operation lies on a *static* combination of procedures at different abstraction levels where problems as detection of conflicts, preservation of invariants, and even order relations (between procedure steps) must be hand coded by end users.

However, in order to obtain a efficient and realistic system that applies hierarchical planning techniques, it is necessary to reduce the amount of work that a control engineer has to do in order to describe a manufacturing domain, and in order to find a program that control its operation.

In addition, the control program obtained must have a sufficient level of detail such that it incorporates every necessary action to carry out a correct execution. Present approaches for the synthesis of operating procedures [1, 14] or machining process [16] do not obtain complete and realistic solutions in this sense because plans obtained are operation procedures intended to be executed by human operators, thus they lack of the necessary level of detail to be considered control programs and executed by a computer, or they are only focused in a small part of the overall manufacturing system.

The approach we present mixes hierarchical knowledge and reasoning aspects with POCL techniques in order to reduce the user effort in the domain description and problem solving phases, and also, in order to obtain complete plans so that they can be considered hierarchical control sequences, which can be easily translated into standard representations of control programs.

Next sections describe in detail our approach. Section 3 is devoted to describe how to represent a manufacturing domain as a hierarchy of agents, and how the knowledge about the behavior and properties of agents is inherited between different abstraction levels. In sections 4 and 5 we show the problems and plans representation of our approach. Section 6 introduces the planning algorithm and the remaining sections show the future work and conclusions about this approach.

3 DOMAIN REPRESENTATION

Our planning architecture conceives a plant as a multi-agent domain (see Figures 2 and 3) where every agent represents the knowledge about the relevant properties and behavior of every factory device. Some aspects of the knowledge representation and planning algorithm here presented are based on a previous system (MACHINE [6]), which uses a non-hierarchical *agent centered* domain model for representing a manufacturing system. In MACHINE the behavior of every agent is described as an automaton and every transition of the automaton is represented as a control activity (Figure 4), using an expressive and rich language in order to represent actions as intervals and, in addition, to manage different kinds of conflicts and interferences which may arise in complex domains like manufacturing systems.

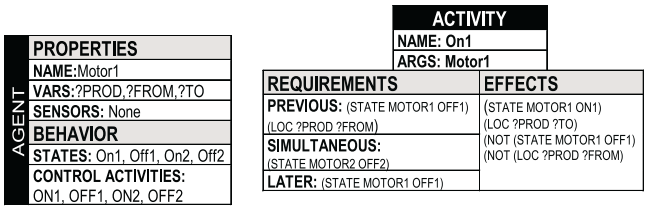

Figure 4. Structure of a primitive Agent.

In this approach, a planning domain is a hierarchy of agents where the root (a "dummy" agent) represents the whole plant (Figure 2), leaf nodes are *primitive* agents corresponding to the field devices of the plant and intermediate nodes are *aggregate* agents, i.e., agents whose structure and behavior are described at higher abstraction levels and which represent a composition of a set of agents at lower levels of abstraction.

Hence, a manufacturing domain is structured at different abstraction levels. Lowest level agents are represented as shown in Figure 4 and its actions are called *primitive activities* intended to be executed by a device.

An aggregate agent is represented as shown in Figure 5. An *aggregate activity* is represented as a primitive one but with an additional property called *expansion*. An example of an aggregate agent and its components can be seen in Figure 6.

The expansion slot of an aggregate activity is used to specify different ways to carry out that activity. These alternative ways are described as a set of different methods [2] where every method is represented by a set of literals (which can be ordered) representing a

[2] Not in the strict sense of HTN.

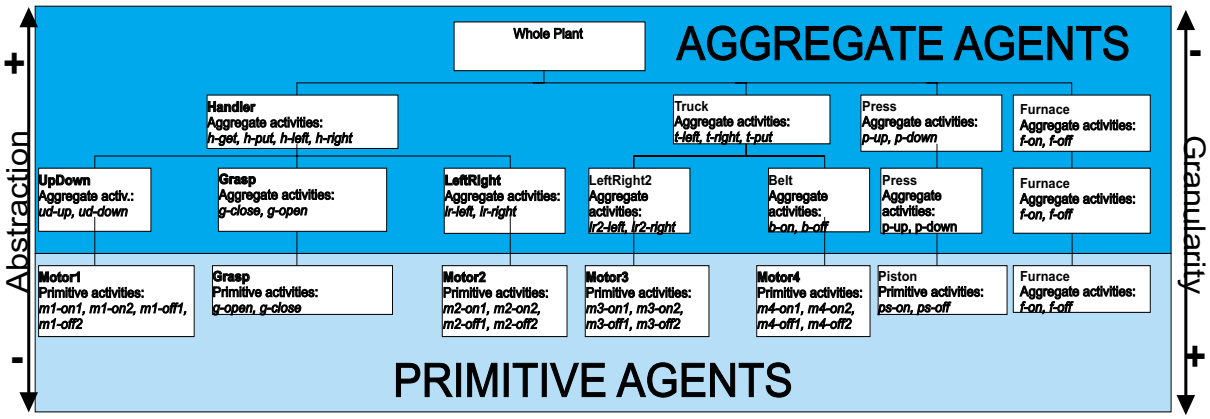

Figure 3. A Hierarchical Domain.

Figure 5. Structure of an Aggregate Agent.

problem to be solved by the agents of the next abstraction level (see Figure 7).

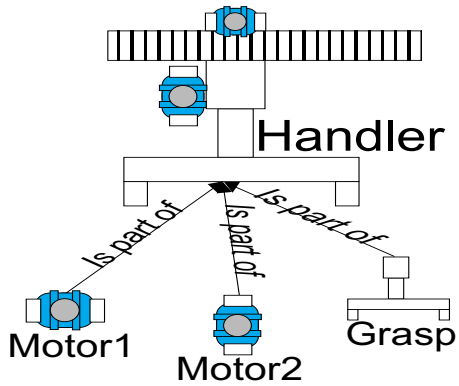

Figure 6. The Handler Aggregate Agent.

In the next section we show how relations between agents and activities at different levels of detail are represented.

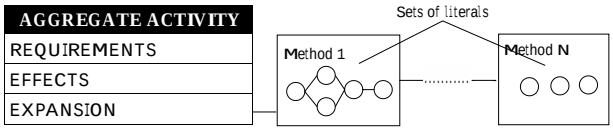

Figure 7. The Expansion Slot of an Aggregate Activity.

3.1 Levels of abstraction and knowledge inheritance

Every aggregate agent of a hierarchical domain is composed of a set of agents at a lower level of abstraction (or higher level of granularity [11]) and it is part of another agent at a higher abstraction level. Hence, properties, activities, states and literals of every agent have an abstraction level associated with them.

Properties and behavior of a given aggregate agent are related with those of its components by means of the *interface* of the aggregate agent. The interface is actually a set of constraints inheritance rules which defines how variables of every component agent inherit their domains and constraints from the ancestor agent (Figure 8).

The aggregate activities of an aggregate agent (its behavior) are represented at a greater *grain size* than the activities of its components. For example, the activities of the agents *UpDown*, *LeftRight* and *Grasp* (see Figure 3) have a lower grain size than the activities of the agent *Handler*. The effects of activity *H-RIGHT* of the agent *Handler* are

(LOC Handler Position2),(STATE Handler Carried).

These literals have a very low level of granularity, and they correspond with the real situation in which the *Handler* agent is up and over the *Truck*, it grasps a piece and it moves to its right. The effects of activity *LR-RIGHT* of the agent *LeftRight* are

(LOC Handler Position2), (LOC LeftRight Position2),(STATE LeftRight Right),

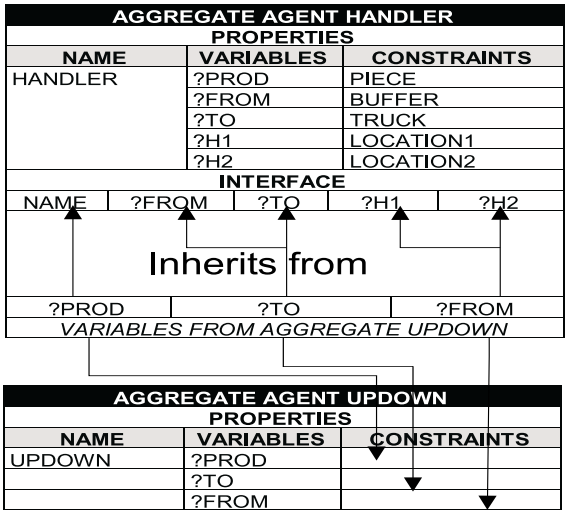

Figure 8. Properties and constraints inheritance.

which have a higher level of granularity and which mean that the *LeftRight* agent is at the same position of the *Handler* agent and, since it is a different agent, its state is moving to its right.

As can be seen, the literals of the requirements and effects of an aggregate activity of an agent may have a different granularity level than the ones of its component agents, and also it is possible that literals of a given granularity level may be different from literals of higher or lower granularity levels.

So, in order to maintain the coherence between different granularity levels activities and literals at different levels must be associated somehow.

The correspondence between a given aggregate activity of an aggregate agent and the activities of its components, and between their different granularity literals, is based on an *activity expansion process* where the aggregate agent's interface and also the expansion slot of the aggregate activity play the key role. The process is described as follows:

- The expansion slot of an aggregate activity of an aggregate agent is represented as a set of methods, where every method is a set of literals at the aggregate granularity level which represents a problem to be solved by the activities of its component agents.
- The literals of these activities have a different granularity, so the activity expansion process applies the rules of the interface of the aggregate agent to every literal in order to articulate the change of granularity. Hence, the interface works as an articulation function [11], between abstraction levels, as follows: every subgoal of a method of an aggregate activity a_g of an aggregate agent g is represented as a literal $(f_1\ x_1\ ...\ x_n)$ where every argument x_i has a domain and constraints at the abstraction level of a_g, then applying the function to this literal will result in a new literal $(f_2\ y_1\ ...\ y_m)$

$$(f_1\ x_1\ ...\ x_n) \xrightarrow{a.f_i} (f_2\ y_1\ ...\ y_m)$$

at a higher level of granularity where every argument y_i is a new argument whose domain and constraints must be consistent with the new granularity level. This literal represents a new subgoal to be solved by the activities of the components of g.

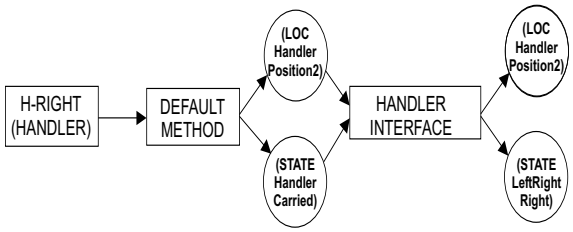

Figure 9. Expansion of Aggregate Activity.

Following this process we obtain a set of literals of lower granularity that represent a problem which must be solved by the activities of the component agents of g by means of a *generative process*.

In addition, the expansion has always at least a *default method* whose literals are the set of effects of the activity. So, a default and *domain independent* expansion process can always be applied for every aggregate activity. Figure 9 shows an example of expansion using a default method. The effects of the activity *H-RIGHT* of the agent *Handler* are the subgoals of the method, and they are mapped by its interface and translated into a set of different literals at a higher level of granularity. These new literals must be solved by the activities of the agents *LeftRigh, UpDown* and *Grasp* at the lower abstraction level.

In addition to this model, the architecture provides a predefined hierarchy of generic classes of aggregate and primitive agents in order to simplify the task of building planning domains, in such a way that every specific agent of the domain is described as an instance of a class of agents (like a *drag & drop* operation).

This model of agents and actions differs of HTN [8] techniques in that expansions (reductions) are not predefined and static substitution rules but a domain independent and dynamic generative process. It also differs from hierarchies of abstraction spaces since sets of operators and literals may be different from an abstraction level to each other. This is because abstraction levels of the proposed hierarchy are based in an increasing semantic granularity instead of in *literal dropping* as in [15, 18].

It possibly looks like models of action in SIPE [21] and OPlan [19], however, their decomposition of actions is defined by the user (plots in SIPE), while in our model the use of an interface between an aggregate and its components and also the expansion of aggregate activities allows for a well defined decomposition by means of the articulation function and default methods, without any participation of the user.

In next sections we will describe the problems and plans representation used in this architecture.

4 PROBLEMS REPRESENTATION

A problem description is a specification of process on products, i.e., a recipe. In our architecture, a problem is represented as an ordered set of literals which represents the process to be carried out by aggregate agents of highest abstraction level. As can be seen at the top of Figure 11 (Step 0) the set of literals received as input by the algorithm represents the ordered set of operations (in SP88 a *procedure*

```
HYBRID (Domain, Level, Agenda, H−Plan)
  If Agenda is Empty
    Then
        If PrimitivePlan? ( H−Plan [Level] )
            Return H−Plan
        Else
          1. RefAlternatives = HowToRefine? (Domain, H−Plan)
          2. While RefAlternatives is not Empty
             2.1. How = Extract (RefAlternatives)
             2.2. REFINE (Domain, How, Level, Agenda, H−Plan)
             2.3. Result = HYBRID (Domain, Level, Agenda, H−Plan)
             2.4. If Result ≠ FAIL
                    Then Return Result
          3. Return FAIL
    Else
        Result = GENERATE (Domain, Level, Agenda, H−Plan)
        If Result = FAIL
            Then Return FAIL
            Else Return HYBRID (Domain, Level, Agenda, H−Plan)
```

Figure 10. The hybrid planning process

recipe) to be carried out by the highest level agents of the manufacturing system represented in Figure 3, that is, a piece must be located in the truck, then it is heated and, finally, it is pressed.

5 PLANS REPRESENTATION

A plan obtained by this architecture is a hierarchy of control sequences (plans) at different levels of granularity, that is, a *hierarchical plan*. Every level in a hierarchical plan is a sequence of control activities to be carried out by agents at the same or higher abstraction level. The last level of the plan is a sequence of primitive control activities (Figure 11).

Every aggregate activity a of an aggregate agent g in a plan level has associated a set of lower level activities of the components of g, which have a causal relation between them and which solves the expansion of a. Hence, a plan can be seen as a modular control program that can be easily translated into standard representations of modular control programs like GRAFCET [12] .

The architecture obtains such a plan following the planning process described in the next section.

6 PLANNING PROCESS

The planning process is a generative and regressive planning algorithm at different levels of detail such that single level plans at higher granularity are refined into lower granularity plans, until no aggregate activities exist on the lowest abstraction level of a hierarchical plan. The generative process is based on a previous non-hierarchical planner [6].

The input to this process is a hierarchical domain and a recipe at the highest abstraction level (a procedure level recipe in SP88, Figure 1). That recipe is preprocessed in order to build a hierarchical plan *H-Plan* with a single abstraction level, containing a set of literals which represent the problem stated by the recipe. As can be seen in Figure 10, the hierarchical domain *Domain*, the initial abstraction level *Level* (the highest one is 1), an initialized task agenda *Agenda* and the initial hierarchical plan *H-Plan* are passed as inputs to the hybrid algorithm. Then it proceeds as follows:

- First, by means of a generative process it obtains a sequence of control activities to be carried out by the highest level agents.
- Second, if the sequence obtained is only composed by primitive activities then the problem is solved. Otherwise, the sequence is *hierarchically refined*, that is, the algorithm expands every aggregate activity, according to its agent interface and its default method or any other method specifically defined, obtaining a new lower level problem.
- Third, the algorithm recursively proceeds to solve the new problem by the agents at the next level.

This is a very general description of the algorithm but, the following describes some important details about the more relevant functions and procedures involved in the algorithm.

HowToRefine?. The result of this function is a list of refinement alternatives of activities which actually represents a heuristic for refining a hierarchical plan *H-Plan*, given a hierarchical domain *Domain*. Depending on the returned heuristic, the behavior of the hybrid algorithm may vary between an ABSTRIPS and an HTN-like behavior. Although many heuristics may exist, the function may return always a default heuristic. This default heuristic is represented by a list with a single element, and its application by the procedure

REFINE results in the expansion of all activities of a given abstraction level, in *H-Plan*, by means of their default method.

REFINE. This procedure applies the above described *activity expansion process* to the aggregate activities of a hierarchical plan *H-Plan*, according to the refinement alternative *How* returned by **Extract**, and taking into account that activities with a lower or equal granularity level than *Level* may be expanded. When **HowToRefine?** returns a unique alternative, representing the default heuristic, it is worthy note that this heuristic applied by **REFINE** turns the **HYBRID** procedure into an *Any Time* hierarchical planning algorithm, because it is able to obtain a plan with no pending subgoals at a given abstraction level (see Figure 11). Additionally, if the refinement process requires it, **REFINE** may introduce a new granularity level in the hierarchical plan, and also may switch to the next abstraction level (increasing *Level*).

GENERATE. This procedure is a generative and regressive planning algorithm based on MACHINE, which is able to represent and reason about actions as intervals and to manage others different kinds of conflicts and interferences which arise in complex domains. However, its features has been extended in order to manage the knowledge inherited by lower level plans from previous abstraction levels in a hierarchical plan (as activities order constraints, established intervals between abstract activities, or the ownership of an activity to the expansion of a more abstract one).

This algorithm solves all of the single-level flaws registered in an agenda *Agenda*, taking into account that, although inherited order relations must be maintained, it is possible to interleave activities belonging to different expansions. These flaws are solved at the next abstraction level of the hierarchical plan. **GENERATE** finally returns a hierarchical plan whith no single-level flaws, but which may contain unexpanded activities.

The algorithm ends when all activities of the lowest abstraction level in *H-Plan* are primitives and there are no pending flaws in *Agenda*. Therefore, the final plan obtained by this algorithm is a hierarchy of control sequences at different granularity levels (See Figure 11 (Step 5)).

As can be seen, the hybrid algorithm here introduced mixes hierarchical and POCL techniques in such a way that the knowledge hierarchy guides the hierarchical reasoning process. Thus, as the hierarchical domain contains a fixed number of abstraction levels, the number of hierarchical refinement levels is also fixed.

6.1 Comparison with other approaches

This approach presents important advantages with respect to previous hierarchical approaches due to the introduction of new issues in the general framework of hierarchical planning [15, 18, 22]. Next we describe the more important ones:

- The default expansion method, defined as the set of effects of an aggregate activity, allows for conceiving the expansion of an activity as a domain independent process. In addition, as it is possible to define alternative expansion methods, the expressiveness of HTN techniques is maintained.
- It is possible to represent a domain as Abstraction Hierarchies [15, 18] and to follow a reasoning process similar to the one used in ABSTRIPS-like approaches. However, the concept of interface and the expansion process here introduced allow for articulating the abstraction levels in a more general way, with a more expressive language and, as we will see, with a reasoning process able to obtain real solutions.
- Unlike HTN [22] techniques, the expansion process of an activity is dynamic and flexible. This means that there not exist a previously fixed reduction of activities, on the contrary, the expansion process of every activity poses a set of lower abstraction goals which have to be dynamically achieved by the agents' activities of the next abstraction level. Thus, the generative process of the hybrid algorithm dynamically stablishes the set of lower level activities that solves the posed problem, in such a way that the expansion process is independent from lower abstraction levels and, therefore, accesible solutions at a given level of the hierarchy are not completely fixed.

In the next section we will discuss some aspects to take into account about the correctness and completeness of this algorithm.

6.2 Correctness and completeness issues

In order to preserve the completeness and correctness of the planning process it is necessary to establish a set of constraints about the way agents and their activities inherit the knowledge of higher abstraction agents. In particular, one of these constraints states that goals with not achieving activity, at a given abstraction level, turn the hierarchical plan unsolvable. Additional completeness constraints are defined in aggregate agents' interfaces and others are "guidelines" on the domain definition in order to represent requirements and effects of activities at different granularity levels. These constraints must be satisfied in the domain elaboration phase of problem solving and must be checked in the action expansion process.

However, although these constraints maintain several established conditions between levels, the Downward Refinement Property (DRP)[15, 22] cannot be satisfied. This property states that the existence of a ground-level solution implies the existence of an abstract-level solution. The contrapositive of this property states that unsolvable conflicts at higher levels always appear in lower ones. Therefore, if this property holds, it is not necessary to refine a plan with an unsolvable conflict. Thus, in this case backtracking between levels is allowed and preserves the algorithm completeness.

Hierarchical planning approaches as [15, 18, 22] are examples about how this property can be satisfied by imposing syntactic constraints in the definition of a domain. However, these constraints reduce the expressiveness of the domain description language.

In abstraction hierarchies [15] levels of abstraction are built from bottom to up by dropping literals from a set of *ground* operators (not reducing the granularity of operators, but relaxing its conditions). In this approach, the set of literals of a given abstraction level contains all literals of the previous level, so lower abstraction levels directly inherit all previous level literals and, hence, every conflict that appears at higher levels also appears in lower levels.

In HTN hierarchies [22], the DRP does not hold in general, but it is possible to establish syntactic constraints in the definition of hierarchical operators in order to satisfy it. However, as can be seen in [22], the syntactic constraints impose that every non-primitive operator has an unique sub-operator which inherits all preconditions and effects of the parent. Therefore, these constraints maintain the grain size of operators between levels.

In our approach, the DRP does not hold because abstraction levels have an increasing size of grain. The states of the automaton that describes the behavior of an aggregate agent are not directly inherited by its components, because they are different states at different

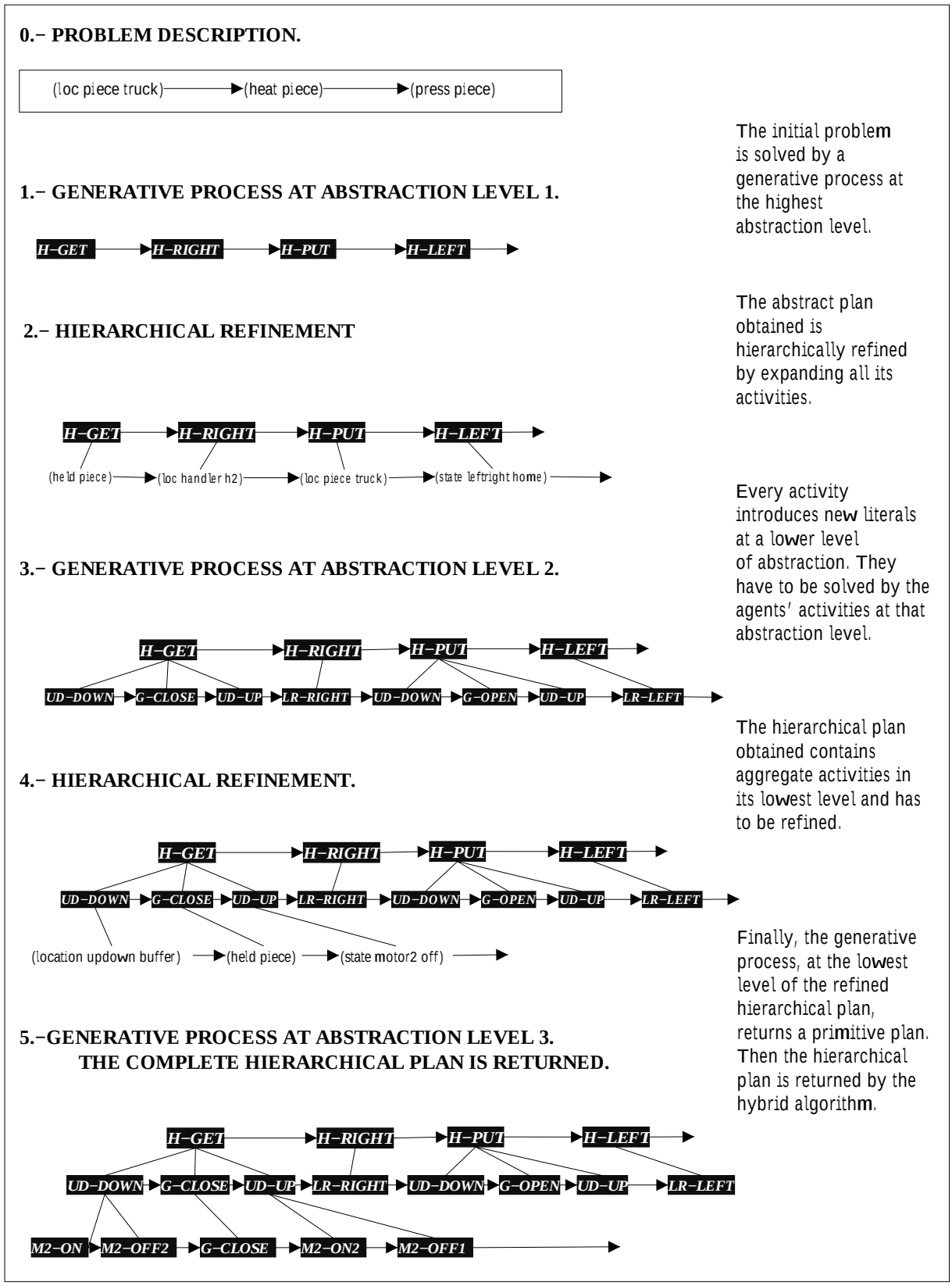

Figure 11. A hierarchical plan obtained by **HYBRID** applying the default refinement heuristic.

granularity levels. This means that there exist literals established in higher levels that disappear in lower levels, so unsolvable conflicts in higher levels may not be inherited by lower levels.

Therefore, we have to face new problems which arise with the application of hybrid planning techniques in a domain representation based on granularity levels:

1. If the DRP does not hold, the algorithm is forced to refine a plan with an unsolvable conflict. The refinement stops when the planner discovers, at a lower level, that the conflict is solved or when the conflict is definitely unsolvable at the lowest level.
2. The activities involved in an unsolvable conflict have to be expanded in order to test the conflict at a lower level. This means that a plan may contain activities and literals at different levels of abstraction (or granularity)
3. The time efficiency of the planner may be reduced if it expands activities up to the lowest level every time that an unsolvable conflict arises.

The second problem has an inmediate solution because the proposed domain representation allows for several levels of abstraction in a plan. For every literal and activity, an unique associated abstraction level always exists, so the harmful effects of the *hierarchical promiscuity* [21] are avoided.

However, a solution to the first and third problem is complex and may be found by extending the heuristics and plans representation currently used in order to manage heterogeneous plans and to offer a correct solution in a reasonable time. At present we are working in this direction but, as can be seen in the next section, the results presented, comparing to the system this approach it is based on, are very promising.

7 EXAMPLES

This section shows the performance of this architecture with respect to the non-hierarchical planner it is based (MACHINE) in the solving of two manufacturing problems.

The layout of the first problem is shown in Figure 12. The problem in this toy plant consists in carry out water from TANK1 to TANK3 and acid from TANK2 to TANK4, but taking into account that they cannot be mixed and that there is only one pump. The plant is represented at two abstraction levels, the circuits are represented in the highest level and the valves in the lowest one. The hierarchical plan obtained is shown in Figure 13 and the performance of the hybrid planning process is shown in Figure 15.

The batch plant of Figure 14 is the configuration of the second problem. In this batch problem there are three types of raw products: an ingredient A, stored into tank T-501, an ingredient B, stored into tank T-505 placed somewhere out of the system, and an ingredient C stored into tank T-504 also out of the system. The hierarchical domain is represented at two granularity levels. The agents of the highest level are represented by aggregating the properties and the behavior of lowest level agents as can be seen in Figure 14

The manufacturing problem *for the lowest hierarchical level* is defined by the following sequence of transformations:

1. STEP 1. Add ingredient B to ingredient A in reactor R-501. During this operation, the mixture must be in agitation.
2. STEP 2. Heat the mixture.
3. STEP 3. Add ingredient C to the mixture maintaining the agitation. During this mixing operation a residual gas is generated which must be evacuated through the scrubber S-501. Part of this gas is condensed and it precipitates at the bottom of S-501. Once the mixing operation ends, this residual liquid must be carried into the external tank T-503.
4. After the addition of ingredient C, the mixture must be cooled and carried into tank T-502.

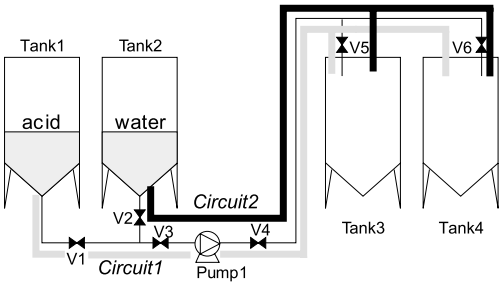

Figure 12. Layout of Problem1.

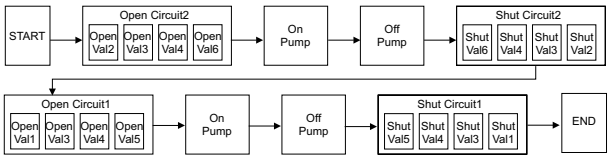

Figure 13. A hierarchical plan for Problem1.

This problem definition is a control recipe specified at phase level. In this case, MACHINE solves the problem exploring about 6000 nodes. However, with this new hierarchical approach the problem may be described as a recipe at operation level whit only one operation: Mix the ingredients in REACTOR. Then the hybrid planning process will obtain first a phase level recipe and, afterwards, the set of control activities of lowest level agents.

These examples show the usefulness of this approach from the

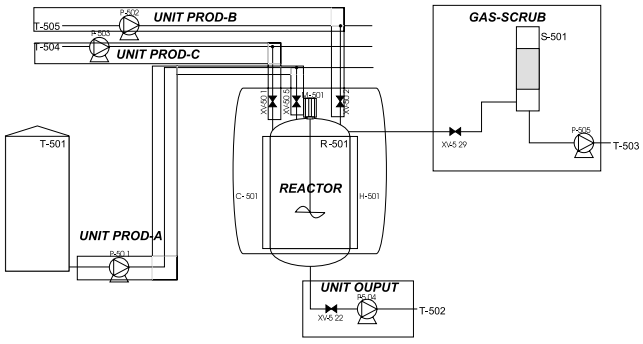

Figure 14. Layout of Problem2.

	EXPLORED NODES	
	MACHINE	HYBRID ARCHITECTURE
PROBLEM 1	400	200
PROBLEM 2	6000	800

Figure 15. Compared search results for Problem1 and Problem2.

view point of a control engineer and its expressiveness and search complexity benefit (see Figure 15). In the next section we describe how to extend this architecture in order to obtain truly realistic solutions.

8 FUTURE WORK

This approach is a step forward in the building of an architecture able to face real world control problems and to obtain fully applicable solutions. However, there are still some steps which must be faced to reach that goal. These steps come up closely related with some of the features that control programs are expected to have, such as robustness or safety.

In a factory, sensory information is a source of uncertainty about the state of the system which forces control engineers to introduce decision and alternative operation steps in a control program, according to the different states of a given sensor. The knowledge representation presented here allows for describing sensors and sensory information but the planning algorithm must still be extended in order to manage the "a priori" uncertainty about states of sensors. This means that the final result of the planning process should be a modular control program with conditional decision structures[9, 17].

This implies that the underlying search space will grow exponentially due to the combinatorial complexity induced by the introduction of conditional branches in a plan. However, it must be said that the effect of this combinatorial explosion can be reduced since the scope of conditional branches can be focused, or isolated, on the basis of a top-down hierarchical process like the one described in this work.

Finally, this approach is being developed within the framework of assisted development of control programs. This means that human operators could interact with the planner and impose their decisions at certain points during the search in a mixed-initiative planning process. The reason for such an interaction is that, in many real world problems, the vast amount of knowledge required for obtaining a solution would produce unrealistically large planning domains and this knowledge can not be completely included. Therefore a planning process must always be open to a possible human interaction which could provide that missing knowledge, what could be seen as a control heuristic to guide the search or to obtain optimal solutions.

The approach presented here follows this direction and it intends to approach these problems in the near future.

9 CONCLUSIONS

We have presented a hybrid architecture which mixes hierarchical planning and POCL techniques, in order to build modular and hierarchical control programs for manufacturing systems.

This architecture is based on a hierarchy of agents by levels of abstraction in such a way that the information granularity of agents, literals and actions increases as the level of abstraction decreases (Figure 3). This representation leads to define different alternatives to existing abstract plan refinement techniques (reduction methods in HTN or plan refinement in ABSTRIPS), the activity expansion process presented is one of them.

The application of hierarchical problem solving techniques results in a lower time and space complexity of this architecture with respect to the system it is based on. However, though the hierarchical domain representation model and planning can reduce the benefit of using these techniques in some cases [2, 3, 4, 10], it has clear advantages from the point of view of computer aided design of control programs.

On the one hand, this approach provides an easy entry-level for end users (control engineers). The hierarchy of agents of a domain accepts SP88 standard descriptions, usually handled by control engineers, so the knowlege can be introduced painlessly.

On the other hand, plans are designed and represented following a top-down process which makes them easier to understand by a human user.

In conclusion, this hierarchical representation and planning process provides a greater efficiency with respect to the non-hierarchical previous version, but, and this is more important for a real world planner, it closes the gap between the planner and their end-users providing a higher degree of integration with them by means of a friendly input level for incorporating knowledge and a more understandable output level.

ACKNOWLEDGEMENTS

This work has been supported by C.I.C.Y.T. under project TAP99-0535-C02-01.

References

[1] R. Aylett, G. Petley, P. Chung, J. Soutter, and A. Rushton, 'Planning and chemical plan operation procedure synthesis: a case study', in *Fourth european conference on planning*, pp. 41–53, (1997).
[2] F. Bacchus and Q. Yang, 'Downward refinement and the efficiency of hierarchical problem solving', *Artificial Intelligence*, **71**, 43–100, (1994).
[3] C. Backstrom and P. Jonsson, 'Planning with abstraction hierarchies can be exponentially less efficient', in *Proc. of IJCAI 95*, pp. 1599–1604, (1995).
[4] R. Bergman and W. Wilke, 'Building and refining abstract planning cases by change of representation language', *JAIR*, **3**, 53–118, (1995).
[5] *PLANET News (Issue No.1)*, eds., S. Biundo and B. Schattenberg, PLANET: European Network on Excellence in AI Planning, 2000.
[6] L. Castillo, J. Fdez-Olivares, and A. González, 'Automatic generation of control sequences for manufacturing systems based on nonlinear planning techniques', *Artificial Intelligence in Engineering*, **4**(1), 15–30, (2000).
[7] L. Castillo, J. Fdez-Olivares, and A. González, 'A three-level knowledge based system for the generation of live and safe petri nets for manufacturing systems', *To appear in Journal of Intelligent Manufacturing*, (2000).
[8] K. Erol, J. Hendler, and D. Nau, 'UMCP: A sound and complete procedure for hierarchical task-network planning', in *AIPS-94*, (1994).
[9] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson, 'An approach to planning with incomplete information', in *Proc. Third. Int. Conf. on Principles of KRR-92*, pp. 115–125, (1992).
[10] F. Giunchiglia, 'Using abstrips abstractions – where do we stand?', *Artificial Intelligence Review*, **13**, 201–213, (1999).
[11] J. Hobbs, 'Granularity', in *IJCAI 85*, pp. 432–435, (1985).
[12] IEC, 'Preparation of function charts for control systems', Technical Report IEC-60848, International Electrotechnical Commission, (1988).
[13] Instrument Society of America (ISA), *Batch control Part 1: models and terminology (SP-88)*, 1995.
[14] I. Klein, P. Jonsson, and C. Backstrom, 'Efficient planning for a miniature assembly line', *Artificial Intelligence in Engineering*, **13**(1), 69–81, (1998).

[15] C. A. Knoblock, *Generating Abstraction Hierarchies*, Kluwer Academic Publishers, 1993.
[16] D. Nau, S. K. Gupta, and W. C. Regli, 'AI planning versus manufacturing-operation planning: A case study', in *IJCAI-95*, pp. 1670–1676, (1995).
[17] L. Pryor and G. Collins, 'Planning for contingencies: A decision based approach', *Journal of Artificial Intelligence Research*, **4**, 287–339, (1996).
[18] D. E. Sacerdoti, 'Planning in a hierarchy of abstraction spaces', *Artificial Intelligence*, **5**, 115–135, (1974).
[19] A. Tate, B. Drabble, and R. Kirby, 'O-PLAN2: An open architecture for command, planning and control', in *Intelligent scheduling*, eds., M. Zweben and M. Fox, Morgan Kaufmann, (1994).
[20] S. Viswanathan, C. Johnsson, R. Srinvivasan, V. Venkatasubramanian, and K.E. Ärzen, 'Automating operating procedure synthesis for batch processes. part I: Knowledge representation and planning framework.', *Computers and Chemical Engineering*, **22**(11), 1673–1685, (1998).
[21] D. E. Wilkins, *Practical planning: Extending the classical AI planning paradigm*, Morgan Kaufmann, 1988.
[22] Q. Yang, *Intelligent Planning. A decomposition and Abstraction Based Approach*, Springer Verlag, 1997.