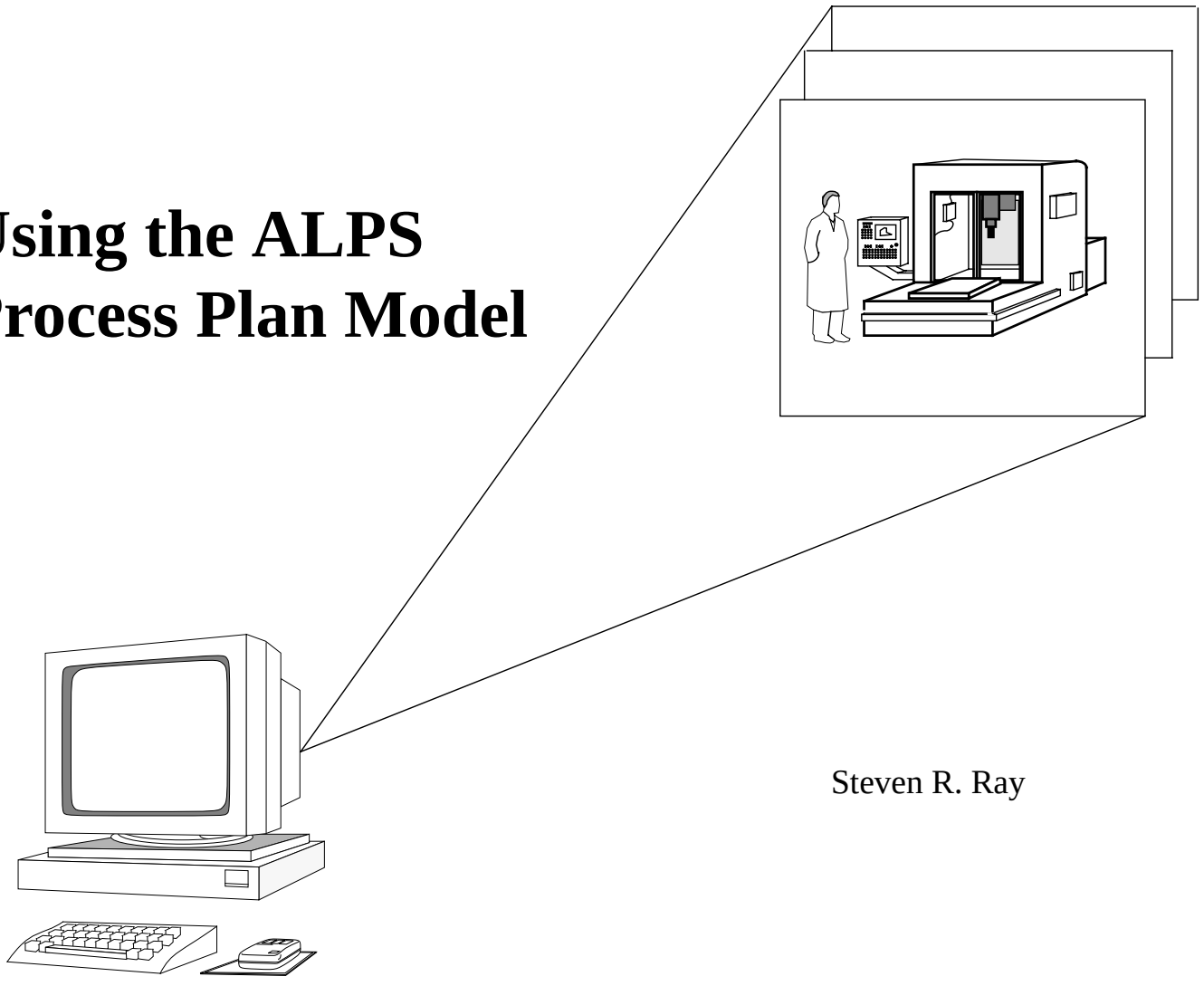
Manufacturing
Systems
Integration

Using the ALPS Process Plan Model



Steven R. Ray

United States Department of Commerce
National Institute of Standards and Technology
Manufacturing Engineering Laboratory
Gaithersburg, MD 20899

Manufacturing
Systems
Integration

Using the ALPS Process Plan Model

Steven R. Ray

Reprint of:

Ray, S.R., "Using the ALPS Process Plan Model," *Proceedings of the ASME Manufacturing International Conference*, Dallas, TX, 1992.

United States Department of Commerce
Robert A. Mosbacher,
Secretary of Commerce

National Institute of Standards and Technology
John W. Lyons, Director

USING THE ALPS PROCESS PLAN MODEL[1]

Steven R. Ray, Ph.D.
Factory Automation Systems Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

ABSTRACT

There is a need in the industrial manufacturing community for a general-purpose model to represent hierarchical process plans with the power to express concepts such as parallelism, alternatives and synchronization. ALPS (A Language for Process Specification) is an experimental model for such a language under development at the National Institute of Standards and Technology (NIST). This paper presents a series of specific examples to demonstrate the intended usage of ALPS. In addition, the ALPS model itself has evolved to version 5. This paper presents ALPS, version 5, and discusses its differences from earlier versions.

1. INTRODUCTION

This paper is intended to provide the reader with some specific guidance on how the ALPS (A Language for Process Specification) model should be used to communicate a variety of process plan constructs. It is assumed that the reader is already familiar with the concepts promoted in the previous publication on ALPS (Catron and Ray, 1991). The development of the ALPS model has been underway for three years, with the current version being version 5. The only other document describing ALPS (Catron and Ray, 1991), documents version 2 of ALPS. Intermediate versions of the model were used in the implementation and testing of a prototype integrated manufacturing environment (Senehi, et al., 1992).

The paper is divided into five sections. The first section provides some background for the work underway on the definition of ALPS. The second section briefly presents the ALPS, version 5 model and describes the specific details which differ from the previously documented version of ALPS (ALPS, version 2). The third section presents examples of ALPS plans to clarify how ALPS should be used in a number of commonly occurring situations. The fourth section discusses the limitations of the current ALPS model, and the fifth section summarizes the paper.

Background

ALPS was designed to serve as a generic model to support process plans used within the discrete-process manufacturing industry. The need for such a generic model became apparent in the context of a series of projects initiated at NIST during the late 1980's addressing various aspects of Computer Integrated Manufacturing (CIM). It was recognized that one of the key elements for factory integration was consensus on the structure of the underlying information used and generated by component systems. One such body of information is the process plan.

Several important research results in the domain of process specification languages influenced the development of ALPS, including the work of (Taha, 1988) and (Pritsker, 1984) on

1. This work was funded in part by the Navy Manufacturing Technology Program.

simulation languages, and (Homem del Mello and Sanderson, 1986) and (Wilkins, 1984) on representations. An earlier publication (Catron and Ray, 1991) provides a more comprehensive review of the impact of these works on ALPS.

ALPS is based upon a directed graph representation[2]. The design goals for ALPS include the support for task decomposition, parallel tasks, synchronization of tasks, alternative task sequences, resource allocation, critical (noninterruptible) task sequences and information manipulation operatives. The rationale and design principles underlying ALPS are covered in (Catron and Ray, 1991). This paper makes no attempt at a complete presentation of ALPS, focussing instead on its use in a limited number of examples.

It is important to recognize that the development of ALPS is taking place in parallel with international standardization efforts to define a standard process plan model (or language). Specifically, work is underway in ISO TC184/SC4 - Process Plan Model; in ISO TC184/SC5 - Global Programming Language Environment / Manufacturing Application Programming Language Environment (MAPLE); and in ISO TC184/SC2 - Robot Programming Languages. Concepts from the ALPS work are being incorporated in the SC4 working group model, and vice versa. Our intention is to test functional concepts using ALPS in a prototype CIM environment and to apply the lessons learned to the standards currently under development. ALPS is intended to be an experimental language, not an international standard.

Chronology

ALPS, version 2, was developed as part of the Manufacturing Data Preparation (MDP) project within the Manufacturing Engineering Laboratory at NIST. The MDP project addressed the generation and access to engineering data in support of CIM (Computer Integrated Manufacturing). MDP ultimately evolved into the Manufacturing Systems Integration (MSI) project, which focussed more on the generation and access to manufacturing information used in the production context, rather than engineering. Thus, while the focus of MSI is on factory management (such as factory status, schedules, order and part tracking), the project requires engineering data, specifically process plans, to drive the system.

2. EVOLUTION OF ALPS – NEW CONCEPTS

The basic structure of the ALPS model has been expanded considerably. The need for many of these extensions became apparent when ALPS was used within a manufacturing integration testbed to drive factory controllers during metal cutting operations (Senehi, et al., 1992). To address the limitations uncovered, the model was modified as shown in the NIAM[3] diagram of Figure 1, which shows the complete class specification for ALPS, version 5. This graphical representation is most useful for communicating the schema (model) to other readers. Briefly, the solid-line circles denote class definitions, dashed-line circles denote attributes of a class, and boxes denote relationships between classes. A companion specification for the schema also exists using the Express language (ISO, 1991a), which provides a machine-readable form. A detailed

2. ALPS does not explicitly prescribe a syntax in the form of printed characters. Rather, it is defined in terms of a conceptual model, which can be implemented either as a database, a memory-resident data structure, or as an ASCII file. This is the approach used within ISO TC184/SC4 in the development of a standard product data model.

3. Nijssen Information Analysis Methodology

presentation of the model has yet to be published. This paper will simply highlight the commonly used portions of the model.

This section of the paper is divided into three subsections. The first discusses some changes from ALPS, version 2, in the overall use and properties of plans. The second addresses changes in

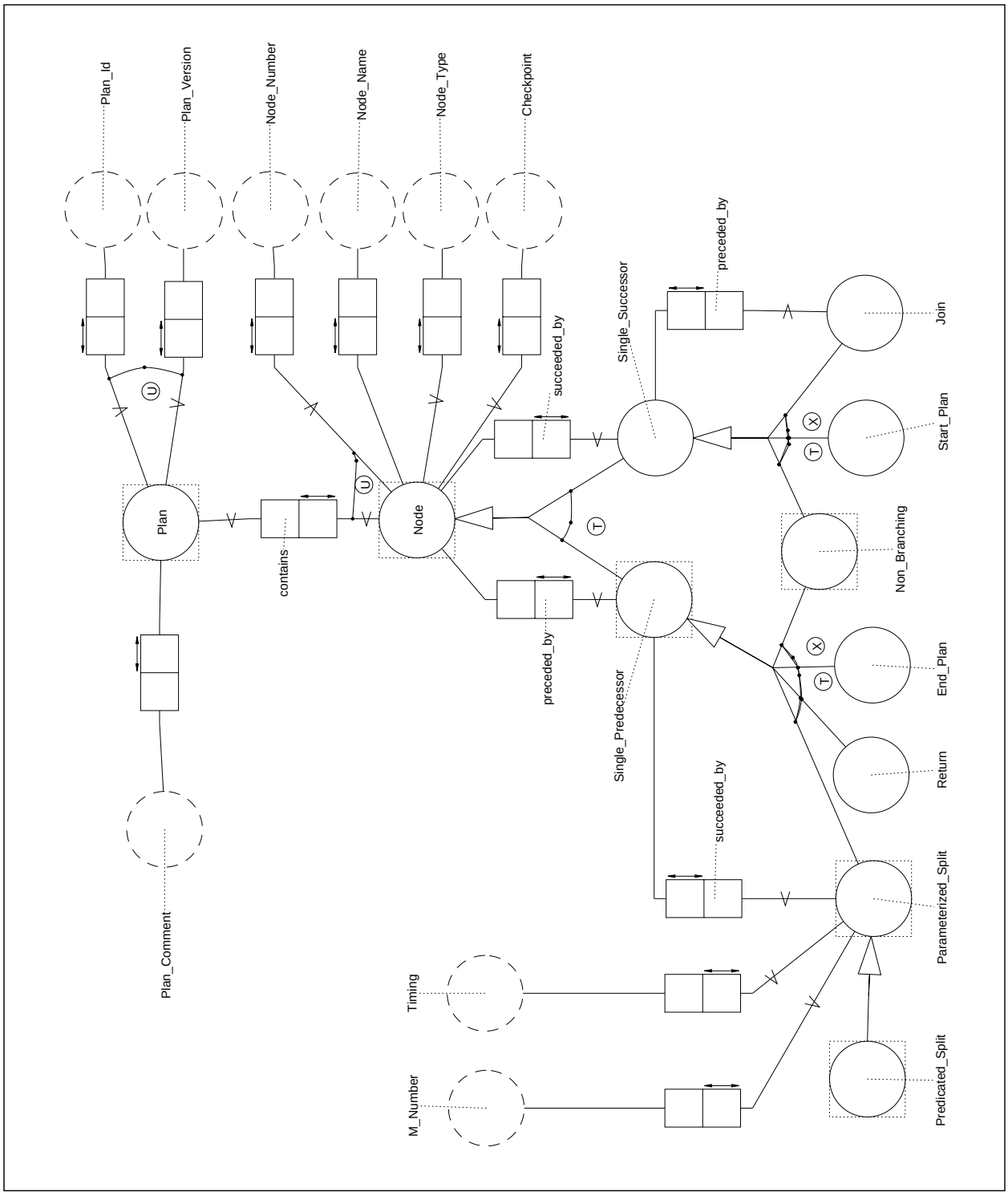


Figure 1. NIAM diagram of ALPS, version 5.4.

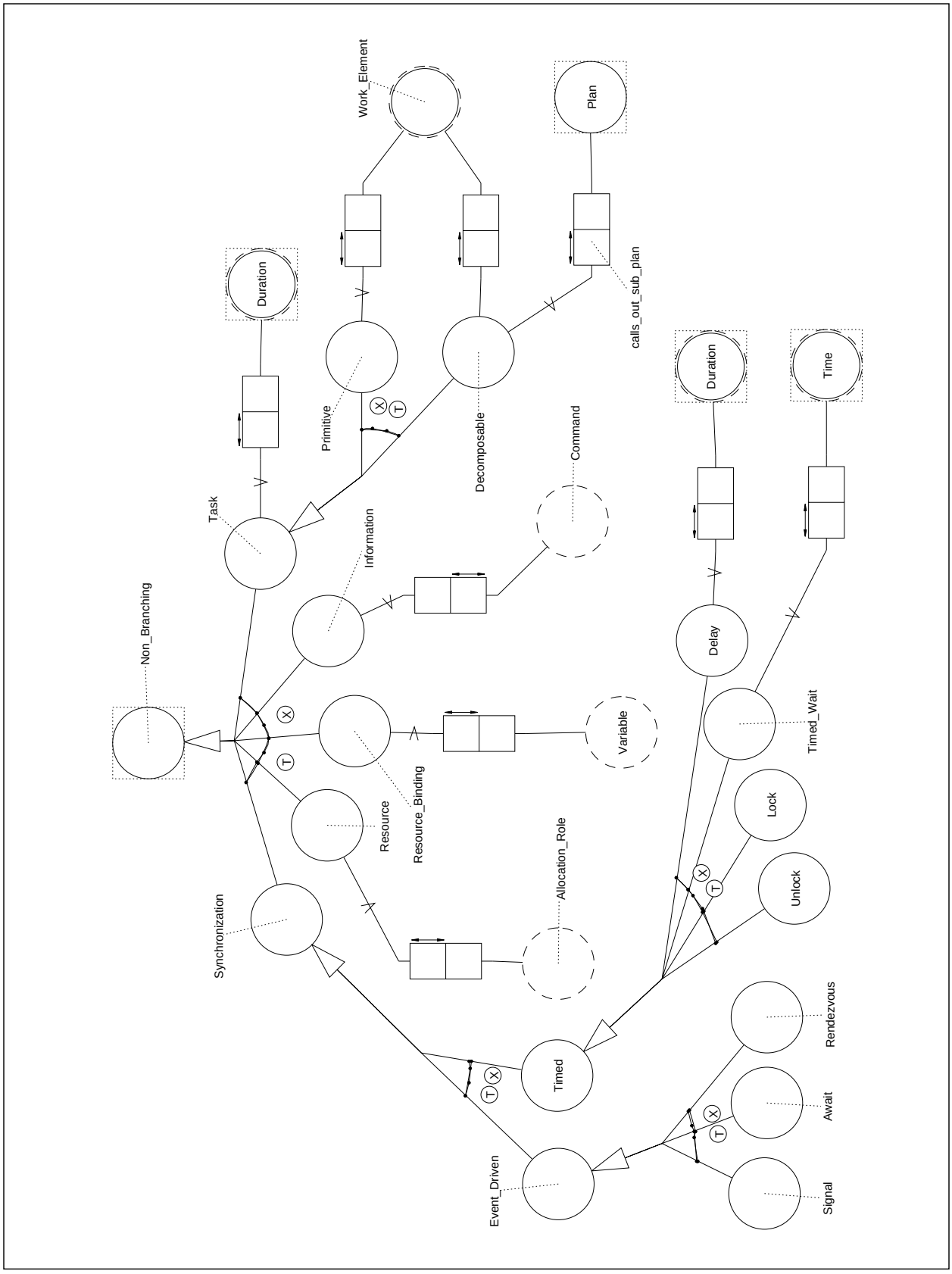
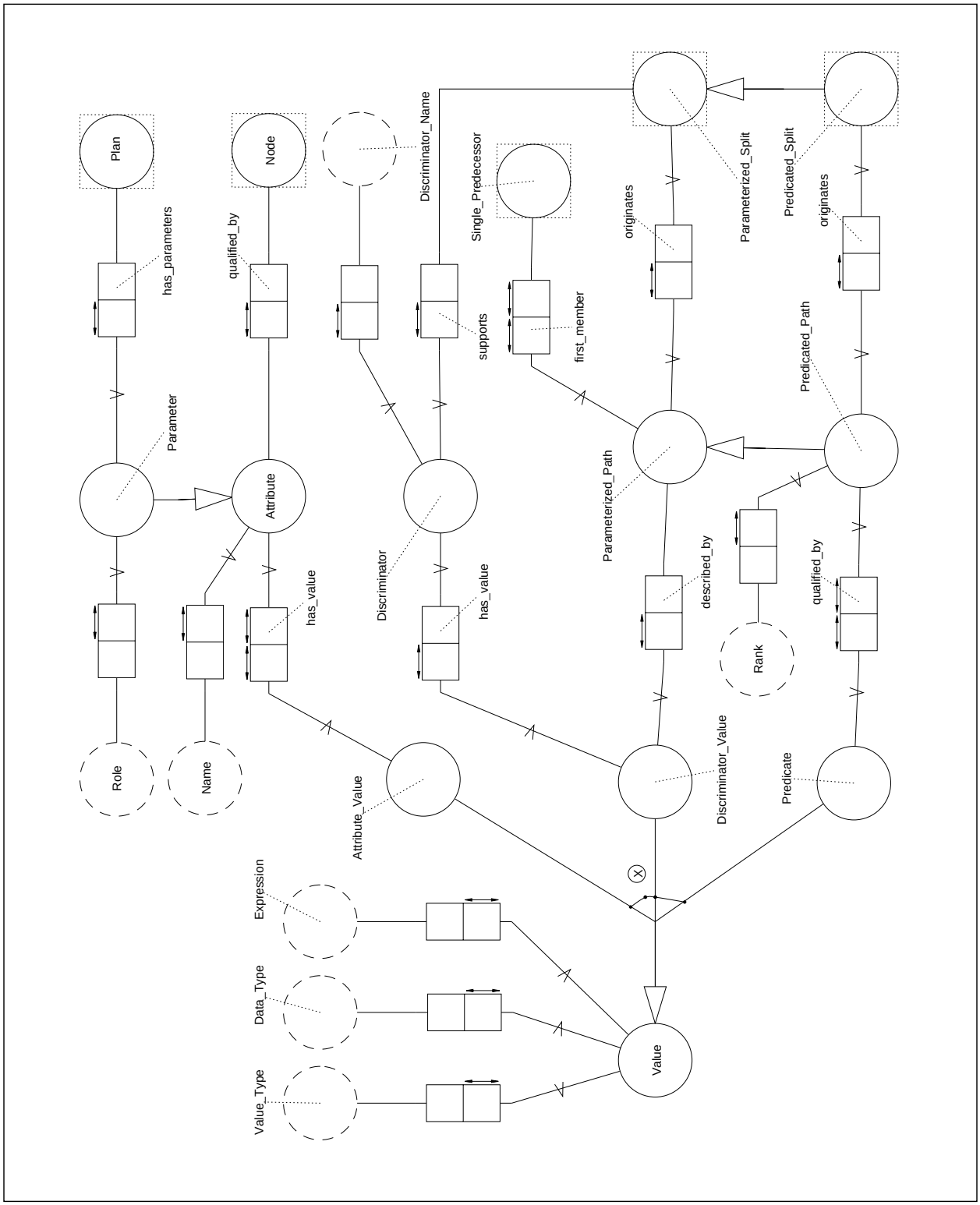


Figure 1. NIAM diagram of ALPS, version 5.4. (continued)

the means of traversing the plans. The third discusses the use of plans by downstream production systems such as schedulers and controllers, and the resulting implications for ALPS.



4. (continued)

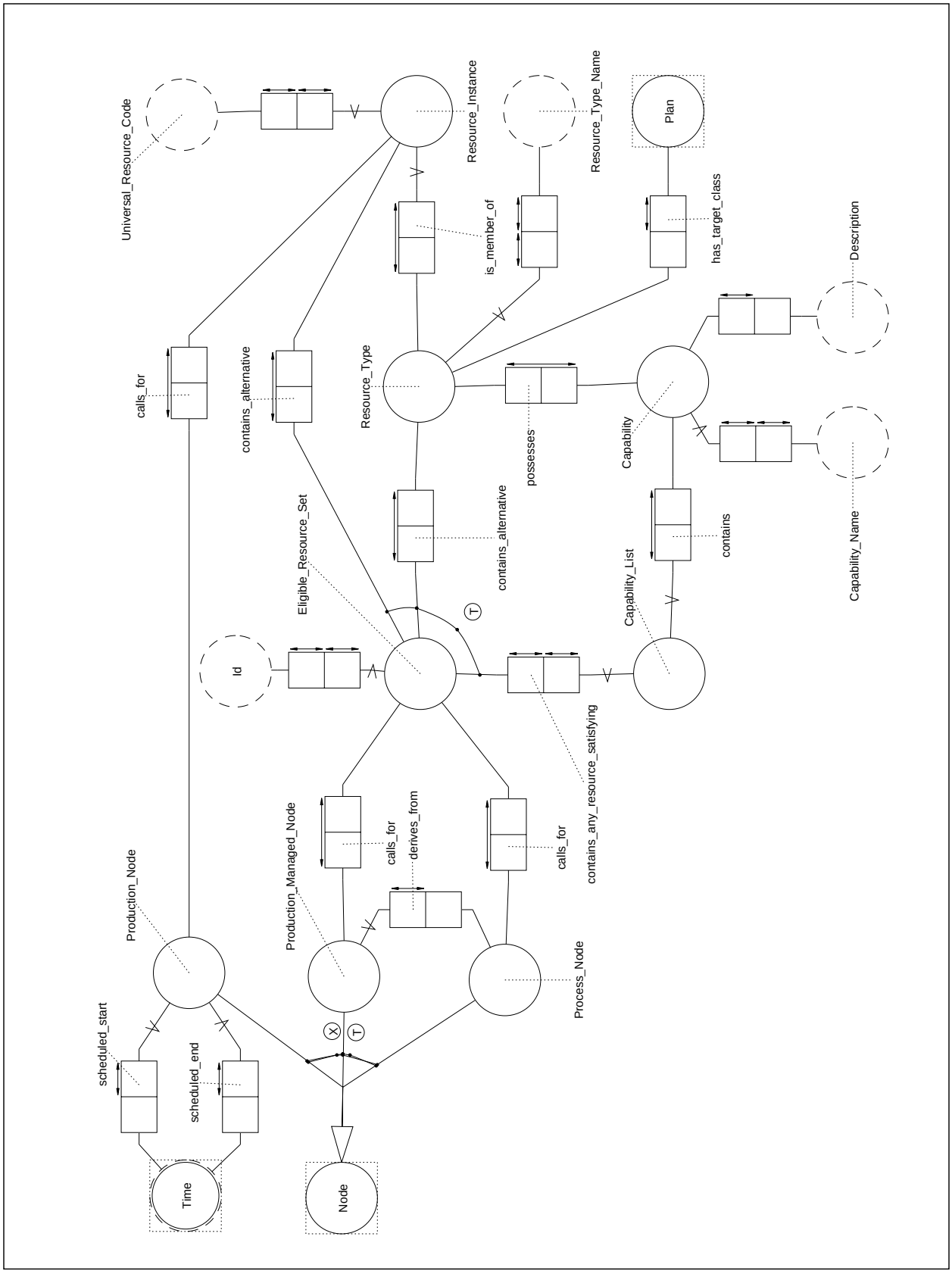


Figure 1. NIAM diagram of ALPS, version 5.4. (continued)

Properties of Plans - Variables, Expressions, and Parameters

One major limitation of ALPS, version 2, was the lack of support for variable values within a plan. This is desirable because it enables the development of parametric process plans which would behave much as a subroutine behaves in a conventional programming language. However, there are even more compelling reasons why variable values *must* be supported within a process plan structure such as ALPS. The most important of these is the need for variables within the Predicates of an alternative branch, that is, a way to express the criteria for choosing one option over another. For a Predicate to have any utility in choosing one of several alternative courses of action, it must evaluate some combination of variables whose values may be numeric, symbolic or boolean, (i.e., it must be a logical expression). Were the Predicate composed entirely of constants, then its outcome would never change, which reduces the Predicate to a tautology.Every ALPS value, whether a variable or a constant, is associated with an expression which carries the actual value. Constants and simple variables are degenerate cases of expressions, where the expression contains no operators, (e.g., "red" and $X, respectively). Other more complex expressions may involve arithmetic or logical operators, (e.g., $X + 5). This brings up the issue of support for generalized expressions, including scoping issues, binding, etc. It is not the intention of the ALPS specification to define a model for expressions. Rather, the ALPS specification simply requires an underlying expression evaluation language which supports the scoping of variables throughout a single plan, but not among plans. That is, variable "x" in plan A is not the same variable as variable "x" in plan B. The use of variables and expressions will become clearer in the examples described later in this paper.

ALPS distinguishes between values and attributes. An attribute is defined as some named property of interest which is associated with a value. A parameter is a subtype of attribute which provides the same functionality as subroutine arguments in a traditional programming language. The role of a parameter can be "in," "out," or "update." The use of parameters thus enables the creation of parametric process plans, which behave much like subroutines with arguments.

Structure of Plans

The handling of branching within plans has changed somewhat from ALPS, version 2. The earlier approach proved cumbersome to use, was not intuitive, and was incapable of representing certain branch conditions. All of these drawbacks have now been addressed. There is still the same categorization of branch points, namely Parametrized and Predicated Split Nodes, each of which possess a timing property of serial, parallel or concurrent[4]. As in ALPS, version 2, Split Nodes support the notion of an M-number, to identify the number of Paths to be taken following the Split. The interpretation of M-number differs slightly for Predicated and Parametrized Split Nodes. For Predicated Splits, the value of the M-number indicates the maximum number of Paths which may be taken. For Parametrized Splits, the M-number specifies the exact number of Paths to be taken. Degenerate cases of the use of M-number include M-number equal to one, which produces an "or" branch point, and M-number equal to "ALL," which produces an "and" branch point. Intermediate values of the M-number produce behavior which cannot be characterized as either "and" or "or" splitting.

Paths. Each branch point (Split Node) in an ALPS plan is associated with two or more Paths which identify the collection of Nodes lying along a given branch. This concept is analogous

4. See Example 3 in the next section for a definition of the timing values.

to that of a "set" of operations in the evolving ISO process plan model. The properties associated with an alternative course of action, for example, are tied to the appropriate Path. An ALPS Path, in turn, is associated with member Nodes via the "first member" relation.

Parametrized Split Nodes. A Parametrized Split Node is used in circumstances where some information is known about the Paths leading from the Node, but there is no explicit criterion for deciding among the Paths. For example, one might know the estimated duration and cost for each of the Paths following the Split. In this case, there would be two discriminators, one for duration and one for cost. The duration discriminator would be associated with several values, each of which is linked to a different Path. The use of a discriminator removes the possibility of "comparing apples and oranges" when deciding among alternative courses of action. All Paths originating from a Parametrized Split Node must support all discriminators associated with that Split (see Example 2).

Predicated Split Nodes. Predicated Split Nodes can be contrasted with Parametrized Split Nodes in that they provide explicit criteria for choosing among the Paths following the Split, by means of Predicates. Each Path is associated with a unique Predicate and its corresponding boolean expression. In addition, each Path is assigned a Rank from 1 to n, where there are n Paths leading from the Split Node. The Rank is used to determine the order of evaluation of the Predicates. There is one exception to this: one Path has the Rank of "ELSE," to provide for default action. This is necessary for the case where all Predicates evaluate to a boolean value of "FALSE."

Plans in the Production Environment

Because of the focus on the use of plans in a production environment within the MSI project, attention has been drawn to providing the necessary functionality within the plan model for downstream systems, notably production management functions including scheduling. A significant effort was made to expand the model to support a variety of mechanisms to reference resources required to execute a given step in a plan. Out of this work, three types of plans emerged which were called process plans, production-managed plans, and production plans. Process plans provide "recipes" for manufacturing a part, and typically contain references to classes of resources required (e.g., 3-axis milling machine). Furthermore, process plans contain no information about *when* tasks are to be performed, but may contain task durations. Production plans, in contrast, make reference to specific, individual resources (e.g., milling-machine S/N 4293), and contain assigned times for the tasks. In general, there would be one production plan per "batch" of parts to be made. Production-managed plans are an intermediate type, where the structure of the final production plan has been defined, but no execution times have been established.

As described above, process plans may call for resources in different ways. As can be seen in Figure 1, production plans identify specific resources individually, while the other types of plans refer to resources indirectly, through eligible-resource-sets. Eligible resource sets may identify resources directly (e.g., machine #5948), by resource class (e.g., three axis milling machine), or by capability (e.g., chamfering).

3. EXAMPLES OF ALPS USAGE

In this section, a number of commonly occurring plan constructs are presented. Two examples are provided which describe alternative courses of action. Then, an example of an

unordered set of tasks is presented. Two methods of handling synchronization of tasks follow. Finally, two examples involving non-preemptable and time-critical task sequences are described. There are clearly many other constructs needed, and which ALPS supports (such as iterative task sequences), but they are not presented here.

A graphical representation of plan fragments is used to best communicate the use of ALPS. Actual implementations would use either an ASCII file representation for the plan[5] or queries to a database repository. In all the following figures, the black arrows denote the precedence of tasks within the plan and the shaded lines denote all other relationships between entities, as defined in the formal schema (Figure 1). The circles represent entities within a populated schema (i.e., instances), and the rounded boxes contain associated properties of those entities. For the sake of clarity, only the relevant properties are shown in each example.

Example 1. *Alternative Paths - Predicated Split*

The example in Figure 2 illustrates the situation where a planner wants to specify two alternative task sequences to be performed, based upon a criterion provided by the planner. In the example, the process plan explicitly provides the criterion to be used in determining which of two branches to execute, namely the availability of machine #5173. Since the M-number associated with the Predicated Split Node equals 1, this branching example behaves like an "or" branch. Path 1 has a Rank of 1, thus its Predicate is evaluated first. If the Predicate for Path 1 is determined to

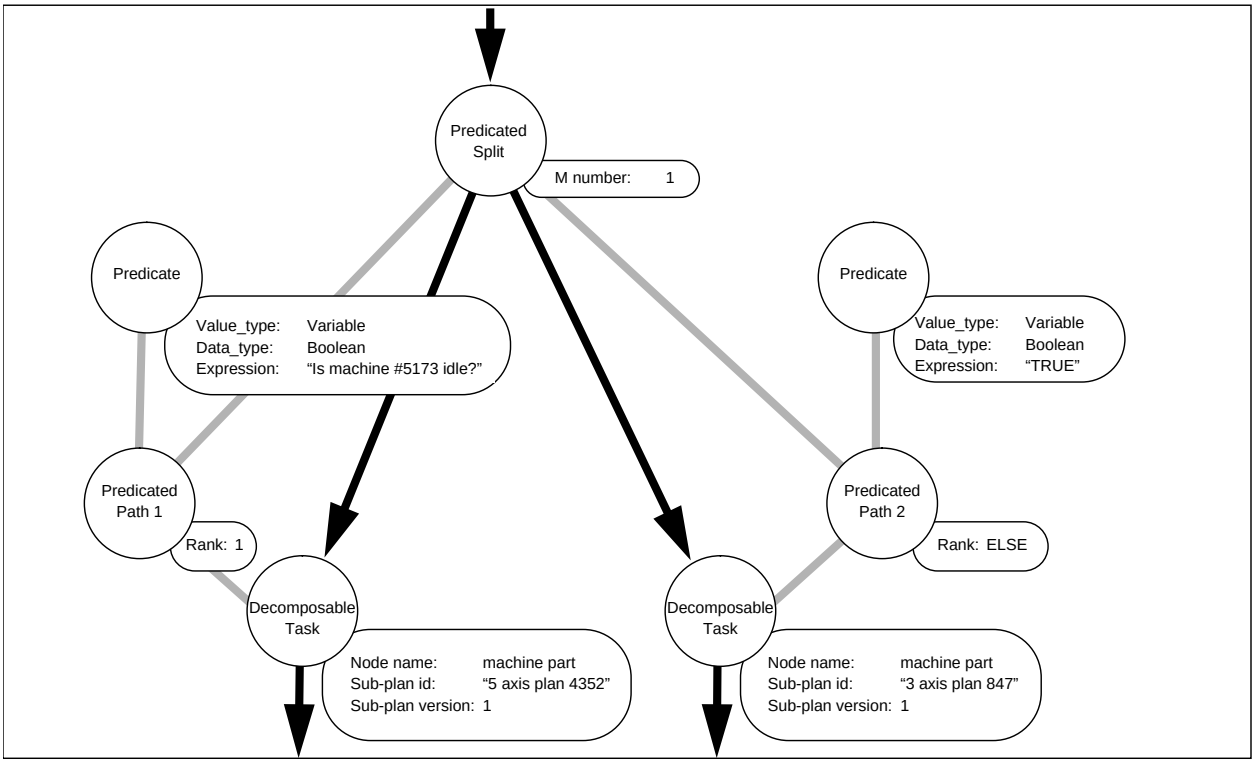


Figure 2. Plan fragment using a Predicated Split.

5. See (ISO, 1991b) for a discussion of the mapping from an Express model specification to an ASCII file format.

be true, then the Decomposable Task referring to the 5 axis plan will be executed. Otherwise, the Decomposable Task referring to the 3 axis plan will be used (the "ELSE" condition).

Example 2. *Alternative Paths- Parametrized Split*

In contrast to the previous example, the example in Figure 3 illustrates the situation where two alternative task sequences exist, but the planner has not provided any explicit criterion to use for choosing the appropriate branch. Rather, properties of each branch are provided (via the discriminator) which can be used in deciding which branch to choose, according to some external business rule. Thus, the reader of this plan would find the two discriminators associated with the Split, one named "duration," the other named "cost." Each discriminator has one associated discriminator value for each associated Path. Thus, there are cost and duration values for Path 1, and cost and duration values for Path 2. In this example one can see that the trade-off is between higher cost with shorter duration, or lower cost with longer duration. Once a decision has been reached (according to criteria external to the plan), the task following the Split Node and associated with the chosen Path is visited. If, for example, time was at a premium, Path 1 would be chosen, so the decomposable task invoking "5 axis plan 4352" would be executed.

Example 3. *Unordered processes*

In the example in Figure 4, use is made of the timing property of Split Nodes, to describe the concurrency of tasks following the Split. A timing value of "serial" implies that the following

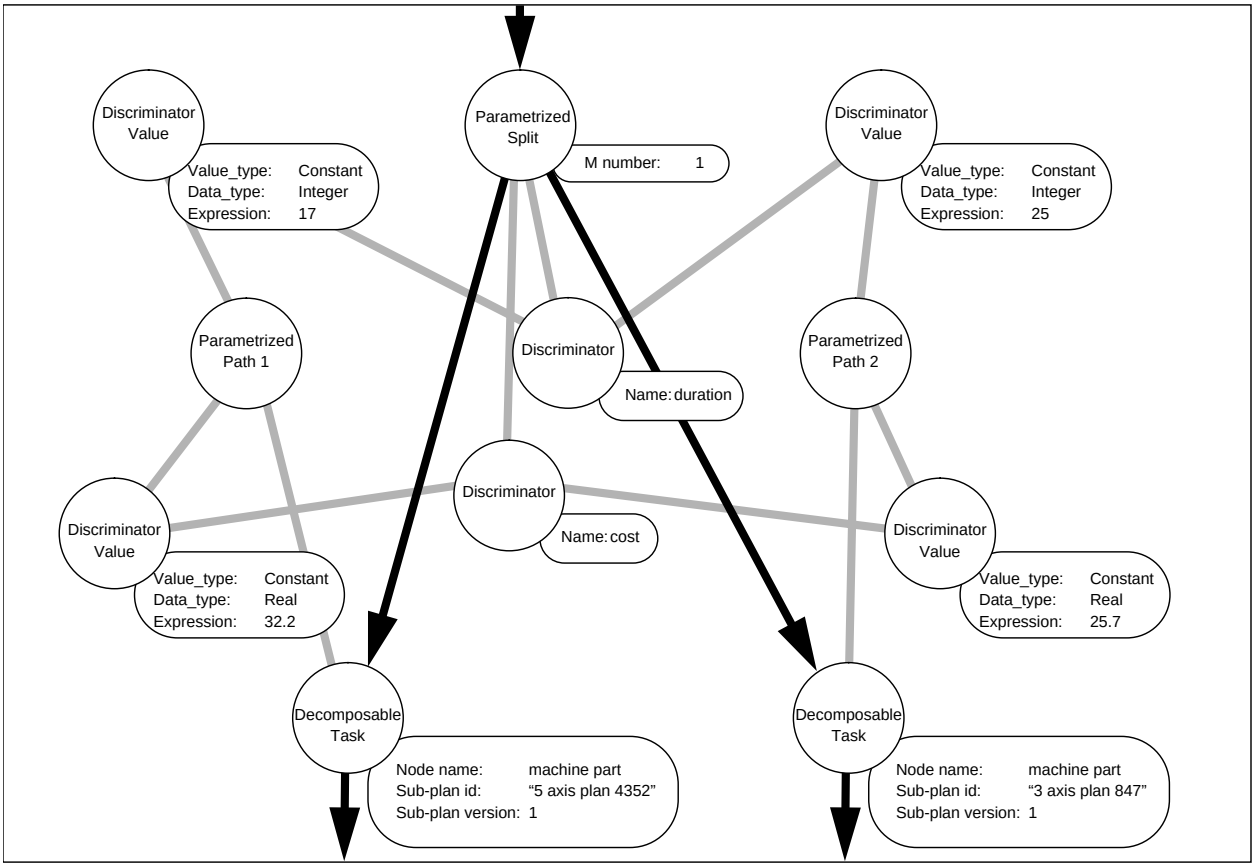


Figure 3. Plan fragment using a Parametrized Split.

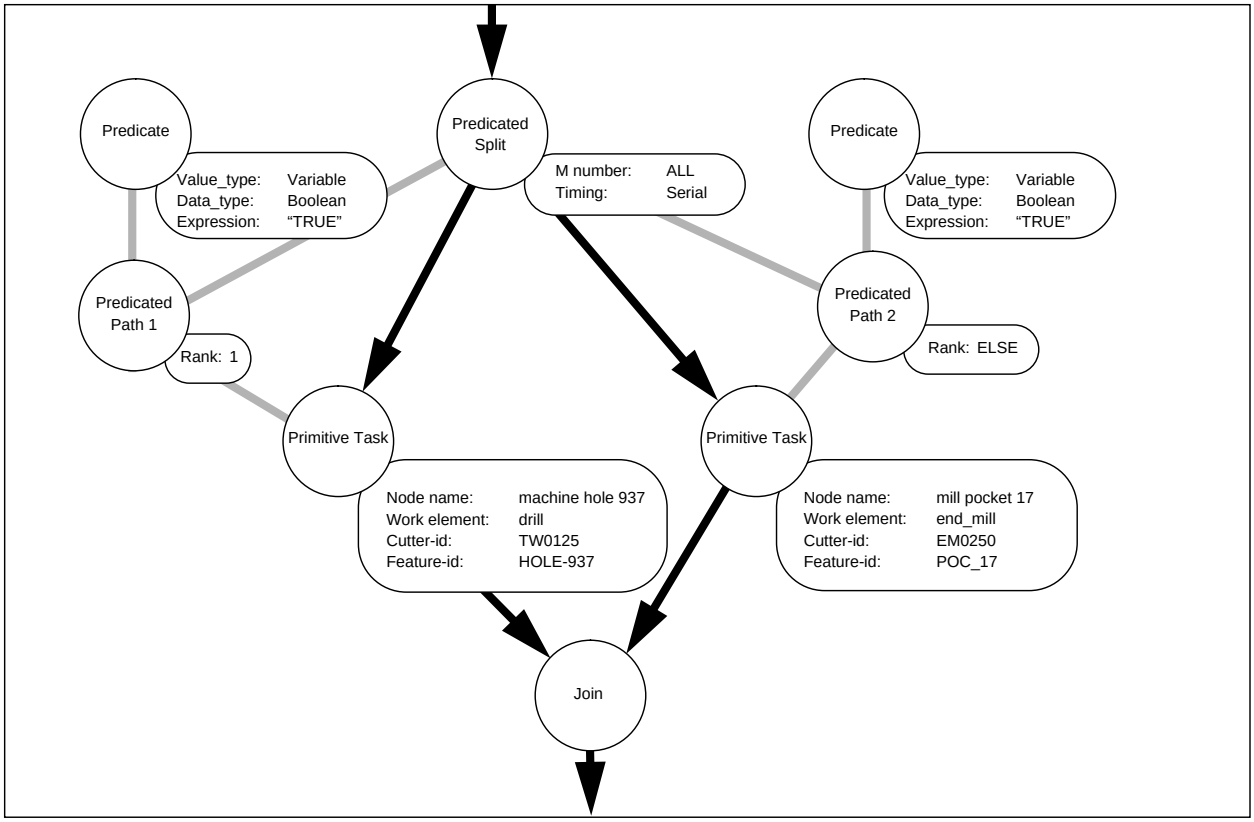


Figure 4. Plan fragment showing unordered tasks.

tasks must be performed one at a time, but the order of execution is not specified. A value of "concurrent" would mean the tasks must be performed simultaneously. A value of "parallel" would mean the tasks may be performed either serially or concurrently - there is no constraint on timing. This example calls for all subsequent tasks to execute, since the M number is "ALL." Furthermore, since the intention is for all tasks to execute, the planner has the choice of using either a Predicated or Parametrized Split. The example uses a Predicated Split. Note that both the Predicates are "TRUE." Finally, this example shows a Join Node which restores the flow of control to a single flow.

Example 4. Synchronized tasks

ALPS supports the notion of plan hierarchies and hierarchical control. There are two methods of synchronizing tasks within ALPS, via supervisory coordination or by means of synchronization semaphores between peers. Figure 5 shows the supervisory synchronization scenario, for the case of a pallet delivery system where a delivery cart and a receiving workstation must both be issued a transfer command at the same time. Thus, it is incumbent upon the agent processing the higher level tasks (the supervisor) to issue the subtasks to its subordinates simultaneously.

This example also demonstrates two additional features of ALPS, version 5: referencing resources and the use of variables within a plan. As shown in the schema (Figure 1), a Node may call for resources directly, by resource type, or by capability. In this example, the cart and the machining workstation are referenced by resource type. During scheduling, the production

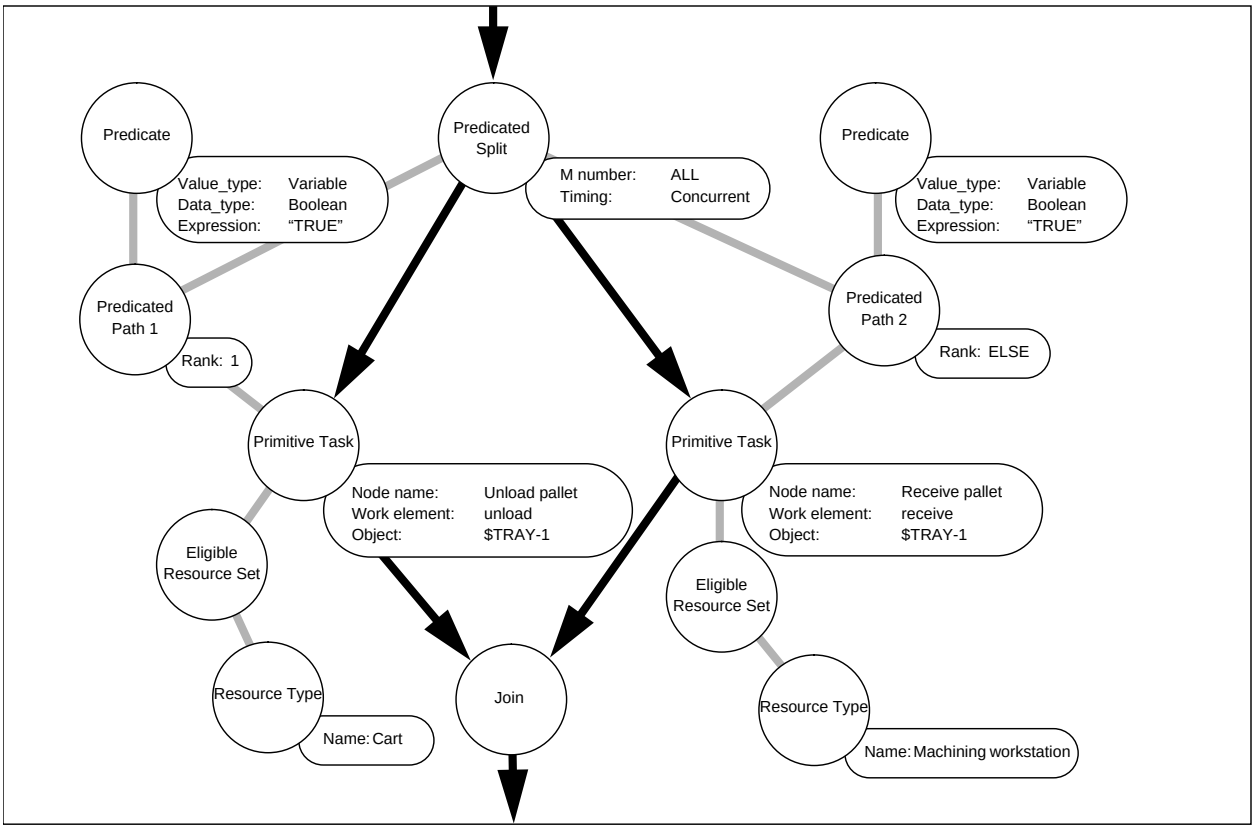


Figure 5. Plan fragment showing supervisory synchronization of tasks.

managed plan is transformed into a production plan and these references are converted to refer to specific instances of a cart and a machining workstation.

The variable $TRAY-1 also appears in this example. This variable must have been bound to a value earlier in the plan, where the value type would have been a particular instance of a tray. This is done by using a Resource-Binding Node[6], which associates a variable name with a reference (direct, by class, or by capability) to a resource.

Using the semaphore mechanism of synchronization allows the subordinates to coordinate their activities without explicitly involving the supervisory level of control. Here, the higher level plan would be as shown in Figure 6a. The Split has a timing type of "parallel" indicating no constraints on the concurrency of the two tasks (at this level of control). The subordinate controllers in this case must also be able to understand ALPS plans, in order to act upon the synchronization semaphores (Rendezvous Nodes) present in the subordinate level plans (Figure 6b). It is important to notice how the synchronization information passes between plans in this example. At the higher level of control, the task for the cart (with Node Name of "unload pallet") binds the attribute "unload event" to the value of the variable $event-23. Similarly, the task for the workstation binds "load event" to the value of $event-23. At the lower level of control, the plan named Cart-485 has an input parameter with the name of "unload event." When the plan is invoked, the value of this input parameter (which is the value of $event-23) is assigned to the

6. See Figure 7 for an example of the use of Resource-Binding Nodes.

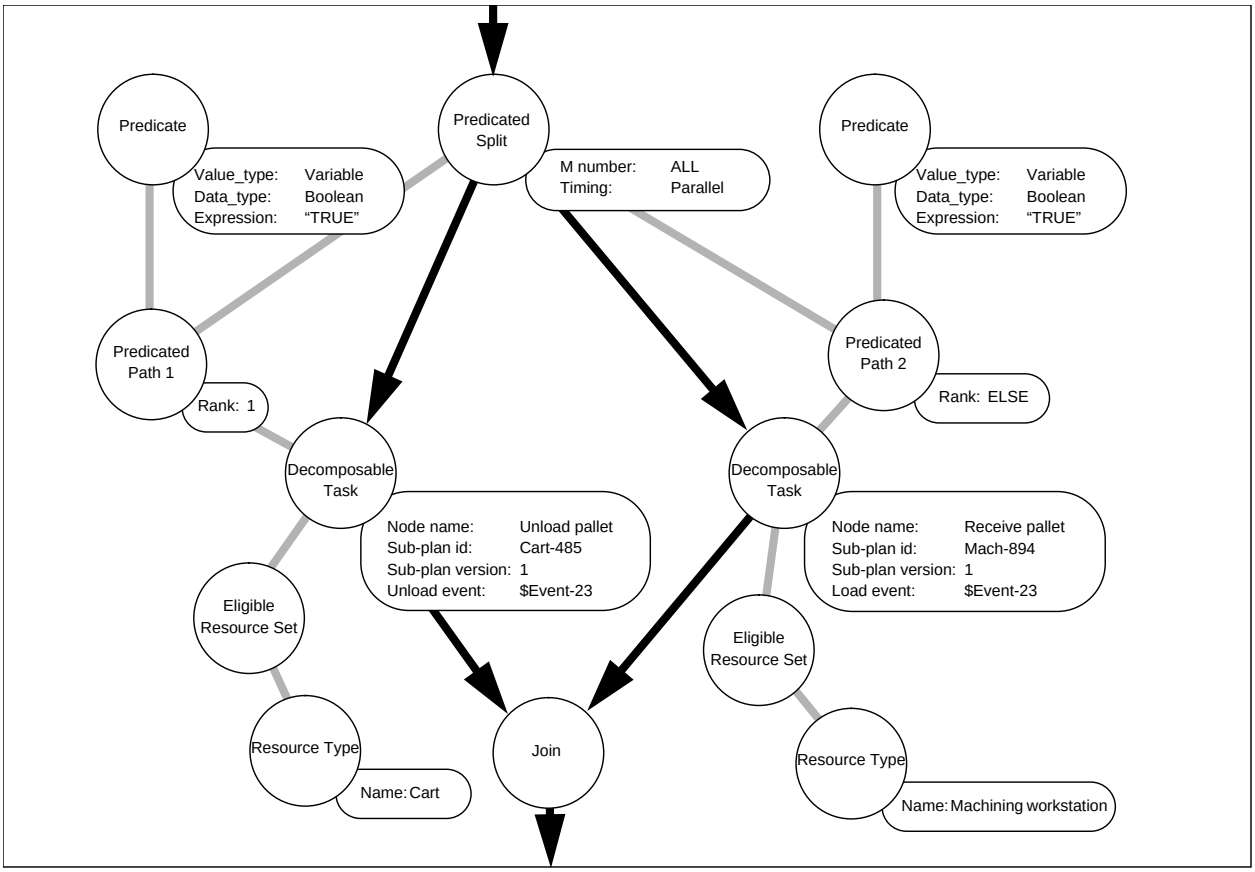


Figure 6a. Plan fragment showing plan relying on subordinate synchronization of tasks.

variable $X, which is then used in the Rendezvous Node. A similar binding takes place in the workstation plan, binding $EVT to the input parameter "load event." This mechanism of parameter passing allows the subordinate level plans to be written independently of one another, without a requirement for agreement on semaphore names. The actual value of $event-23 can be determined at either scheduling or execution time.

Example 5. *Non-preemptable Sequences*

This example (Figure 7) shows how one can describe a sequence of tasks which must be executed without interleaving them with other competing tasks. The example describes the steps which a tool delivery robot must follow to get a tool from a crib to a destination work center. It must move to the tool crib, pick up the tool, move to the destination work center, and put down the tool. Once the robot has picked up the tool, it must complete the sequence for one delivery before beginning the second delivery, presuming the robot can only hold one tool.[7] The Resource-Binding Node binds the variable $R to a specific instance of a robot (to be done at scheduling time). It is that particular robot which is then allocated over the three succeeding tasks, such that the robot cannot be used for any other purpose. Note that this scenario does not preclude pausing the robot during its activities, in contrast to the next example.

7. In contrast, there are other circumstances where it might actually be desirable to interleave two requests for some service in order to optimize performance

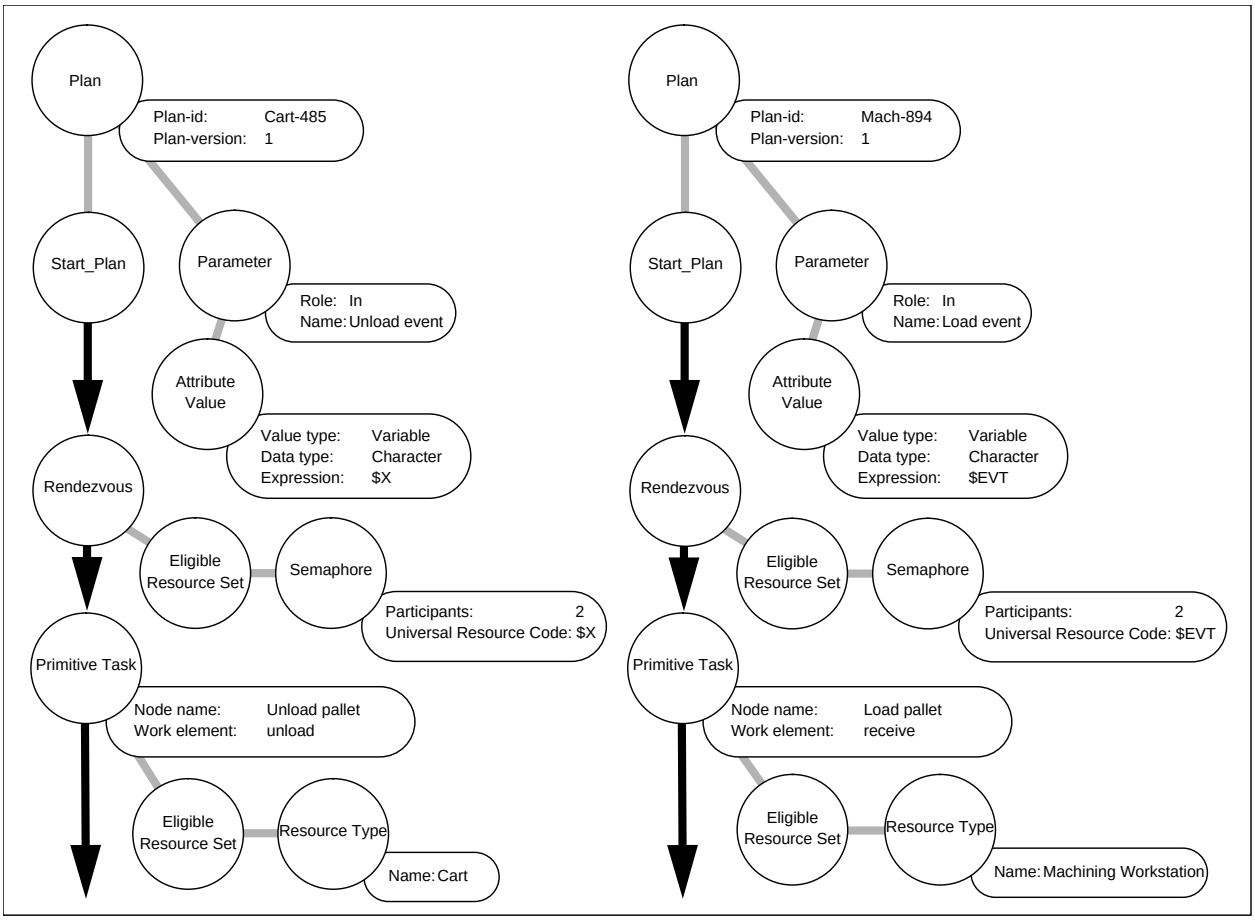


Figure 6b. Portions of the subordinate plans supporting the plan of Figure 6a.

Example 6. Time Critical Sequences

Figure 8 shows a sequence of tasks similar to those in the last example, except that this time there is no locking of resources. Rather, the constraint is that the sequence of tasks be performed without interruption. This behavior is accomplished by using the Checkpoint attribute of a Node. In all other examples (including the previous one) the Checkpoint attribute of every Node had the default value of "CHECKPOINT" and was not shown, for clarity. In this example, the Checkpoint attribute of some Nodes is "null," meaning that activity must proceed uninterrupted following the completion of these Nodes. This concept is similar to identifying critical sections of computer code. In other words, once you start to mix the epoxy, you must finish mixing, take it to the application area and apply it, without interruption. Since the application step has a Checkpoint value of "CHECKPOINT," it is permissible to pause at the completion of this step.

4. LIMITATIONS OF ALPS

ALPS still has a number of limitations, the most important of which are discussed below.

Complex constraints

ALPS does not possess the power to express arbitrary constraints on relationships between Nodes in a plan. Some simple constraints are supported, such as the precedence of tasks, and the

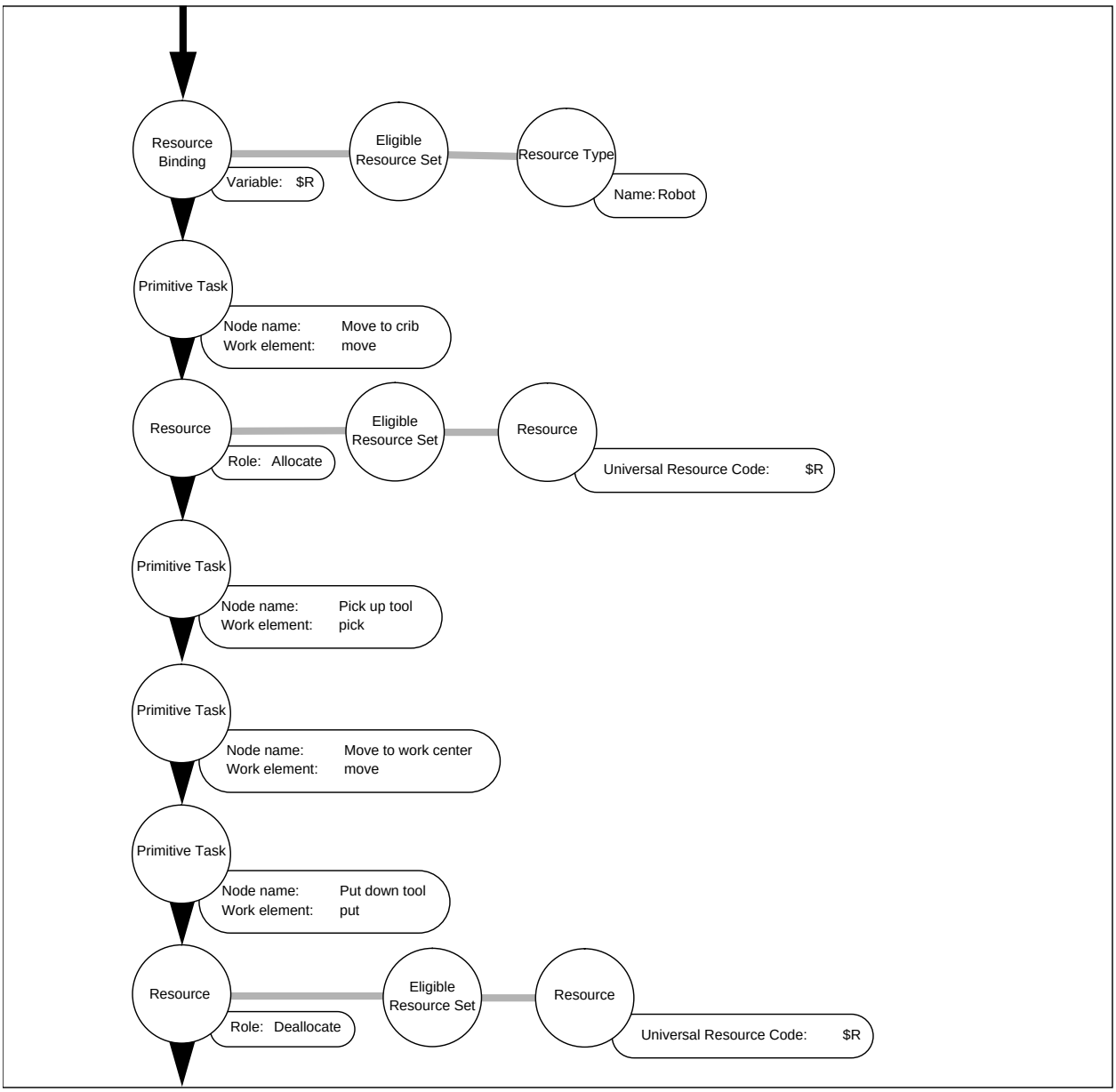


Figure 7. Plan fragment demonstrating a non-preemptable task sequence.

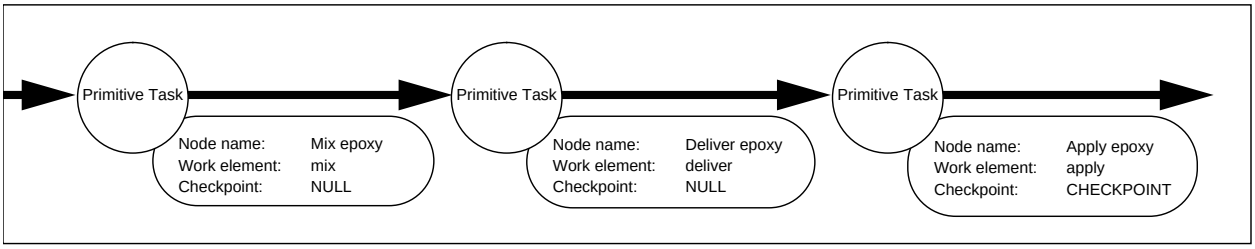


Figure 8. Plan fragment demonstrating a time critical task sequence.

specification of decision criteria when confronting alternative courses of action. In general, however, one would like to be able to define more complex relationships. In Example 5, one should be able to specify an upper limit on the time interval between mixing the epoxy, and applying it. These two steps are not consecutive in the plan, and ALPS provides no way to express such a constraint. Instead, a user of ALPS currently has no choice but to overspecify the constraint by mandating that the entire sequence of tasks be executed without interruption. Thus, there is a need for an embedded, general purpose constraint language suitable for use in plans. This topic is a research area in its own right, and will be addressed if circumstances allow.

The presence of constraints within production plans does raise an important philosophical issue, namely, how much information should a plan contain? At one extreme, a plan could contain all constraints about how to accomplish some goal, leaving it up to some downstream agent to construct a sequence of tasks. At the other extreme, the plan could simply depict the final decision on what tasks to carry out, (i.e., just the "how" but none of the "why"). While a limited amount of flexibility is desirable within a plan, the purpose of ALPS was to focus primarily on specifying the task sequence itself, without much of the underlying rationale. Nevertheless, an embedded constraint language is still necessary in situations such as in Example 5.

Deferred evaluation of expressions

ALPS lacks a way to specify when a given expression should be evaluated. This will become important, for example, when using Predicated Split Nodes to describe alternative courses of action. Some decisions can and should be made at scheduling time. For example, when choosing which of two machine classes to use to process a part, one might use machine availability as the criterion, as in Example 1. However, some alternative branches might be based upon real-time sensory data, such as iterative processes which continue until some condition is reached. It is important to be able to communicate whether a given Predicate should be evaluated during scheduling or during execution. A general solution to this problem is difficult to formulate, since the notion of when an expression is to be evaluated is somewhat domain dependent. One possible approach is to define distinct "stages" for plans in a given domain. Another possibility is that the evaluation time is determined by the hierarchical level of the process plan (e.g., expressions in the lowest level plans are evaluated during execution, expressions in all other plans are evaluated during scheduling). Again, this issue will be addressed if the opportunity arises.

5. SUMMARY

The ALPS schema facilitates communication between manufacturing control systems. It is a general purpose vehicle for any discrete-process manufacturing environment. The database implementation serves as an integrating tool, allowing convenient sharing of processing information.

The ALPS activity supports manufacturing integration work being done at NIST and in the development of the ISO STEP process plan schema. The ALPS schema is intended to be one of several schemas necessary for complete integration of manufacturing information, including a part definition schema, a facility resource schema, and a production management schema.

Future work will focus on refining ALPS and testing it in a prototype integrated production management and control system. The resulting experience will then be transferred to the relevant

standards committees. The challenge of reaching international consensus on a standard plan representation still remains.

REFERENCES

Catron, B., and Ray. S., 1991, "ALPS - A Language for Process Specification," *International Journal of Computer Integrated Manufacturing*, Volume 4, Number 2, pp. 105-113.

Homem de Mello, L.S., and Sanderson, A.C., 1986, "AND/OR Graph Representation of Assembly Plans," *Proceedings of AAAI-86*, Vol. 2, pp. 1113-1119.

Pristker, A.A.B., 1984, *Introduction to Simulation and SLAM II*, John Wiley & Sons, New York, NY.

Senehi, M.L., Wallace, S., and Luce, M., 1992, "An Architecture for Manufacturing Systems Integration," *Proceedings of the Manufacturing International Conference.*

Taha, H.A., 1988, "The SIMNET Simulation Language," *Computers in Industrial Engineering*, Vol. 14. No. 3, pp. 281-295.

ISO 10303-11, 1991a, "EXPRESS Language Reference Manual," ISO TC184/SC4/WG5: Document N14 (Release Draft) (Available from the IGES/PDES/STEP Administration Office, National Institute of Standards and Technology, Building 220, Room A127, Gaithersburg, MD 20899.)

ISO CD 10303-21, 1991b, "Product Data Representation and Exchange - Part 21: Clear Text Encoding of the Exchange Structure," ISO TC 184/SC4 N78 (Committee Draft) (Available from the IGES/PDES/STEP Administration Office, National Institute of Standards and Technology, Building 220, Room A127, Gaithersburg, MD 20899.)

Wilkins, D.E., 1984, "Domain-independend Planning: Representation and Plan Generation," *Artificial Intelligence*, Volume 22, pp. 269-301.