INT. J. COMPUTER INTEGRATED MANUFACTURING, 1991, VOL. 4, NO. 2, 105-113

ALPS - A Language for Process Specification

Bryan A. Catron
Steven R. Ray
Factory Automation Systems Division
National Institute of Standards and Technology

Abstract

This paper describes research identifying information models to facilitate process specification and to transfer this information to process control. A conceptual schema is introduced as a means to bridge the gap between process planning requirements and production control requirements. The result is a process specification language based on directed graph notation which allows full specification of parallel activities, event synchronization, alternative processes, resource management, and task decomposition.

I. Introduction

A crucial step in successful automated manufacturing is the smooth transfer of information between systems in the manufacturing environment. Current research at the Automated Manufacturing Research Facility (AMRF) of the National Institute of Standards and Technology (formerly the National Bureau of Standards) is addressing integration issues for streamlined data integration to support flexible discrete manufacturing.

The AMRF was established in 1981 to serve as a test bed facility to support research in measurement techniques and computer interface standards that are required for automated machining of parts in small lot sizes. The primary thrust of the project was to establish clear interface specifications and support modular structures to allow plug-compatibility between systems. This plug-compatibility allows both a flexible manufacturing environment and offers the capability of incremental automation in existing facilities.

The AMRF is built around the concept of hierarchical control, where high level commands are decomposed into sequences of simpler commands at the next lower level in the hierarchy. The simpler commands are in turn decomposed at yet lower levels. Well-defined protocols have been established to allow command and status information to flow upwards and downwards in the hierarchy. The bulk of data transfer (such as process plans and part models) occurs directly with a distributed data administration system. A mechanism has been implemented to allow any controller in the AMRF to request or store information in a generic way, regardless of which database is being used to hold that information. The adoption of such an architecture avoids many potential information bottlenecks. Further, by adopting a hierarchical approach, the complexity of a task is reduced to a manageable level for each controller in the hierarchy. More details on the AMRF can be found in (Simpson 1982, Furlani 1983, Hocken 1983, McLean 1983, McLean 1985, Nanzetta 1984).

The long-term goal of process planning research in the AMRF is to develop the capabilities to allow real-time, reactive planning. A crucial step to enable real-time planning is the definition of an architecture which will allow the smooth transfer of information among all systems in the manufacturing environment. This architecture is under development within the AMRF, and is based upon a set of conceptual information models implemented as object-oriented databases (Ray 1989).

There are two schemas under development which are relevant to this paper: the Plan Formulation schema and the ALPS schema. Further details on the Plan Formulation schema and its relationship to the ALPS schema can be found in (Ray 1989.) Among other things, the Plan Formulation schema describes the relationships between planning goals, processes to achieve these goals, and resources required by the processes. The Plan Formulation schema is specifically designed to address the information needs during process planning. The schema is not particularly suitable for use by downstream systems, such as schedulers, resource managers, and execution systems. A scheduling system needs information such as the sequencing requirements of processes, trade-offs between alternative approaches, resource commitments, costs, and delivery times. For example, a scheduler is not interested specifically in why certain processes are required, but in just how long a process takes, how much it costs, and what resources are needed. Furthermore, the tasks identified in the Plan Formulation schema are not stratified according to a control hierarchy. Thus, the data is not in a form convenient to other systems. There is a need to re-express this data in terms which are compatible with the rest of the manufacturing environment. This re-expression is specified by the ALPS schema. This paper presents a language specification (ALPS) and the underlying schema which meet the above communication needs.

The paper is divided into eight sections. Section I describes the AMRF research environment and the goals of process planning research in the AMRF, which motivated much of the work presented in this paper. Section II explains the need for a generic process specification language in manufacturing. Section III briefly identifies related work. Section IV defines and discusses ALPS (A Language for Process Specification), including definitions of all node classes and the underlying conceptual schema, which is formally defined in the Appendix. Section V offers conclusions and a discussion of future research directions.

II. The Need for ALPS

ALPS is used as an interface between process planning and production control, and internally between production control processes (Figure 1). In the AMRF, the design of flexible controllers calls for them to be completely driven by process plans. This allows "re-programming" a controller by selecting a new process plan. For this type of automated, data-driven system, a complete description of processing is required in order to manufacture the part. A process specification language can be used to fully communicate manufacturing process steps.

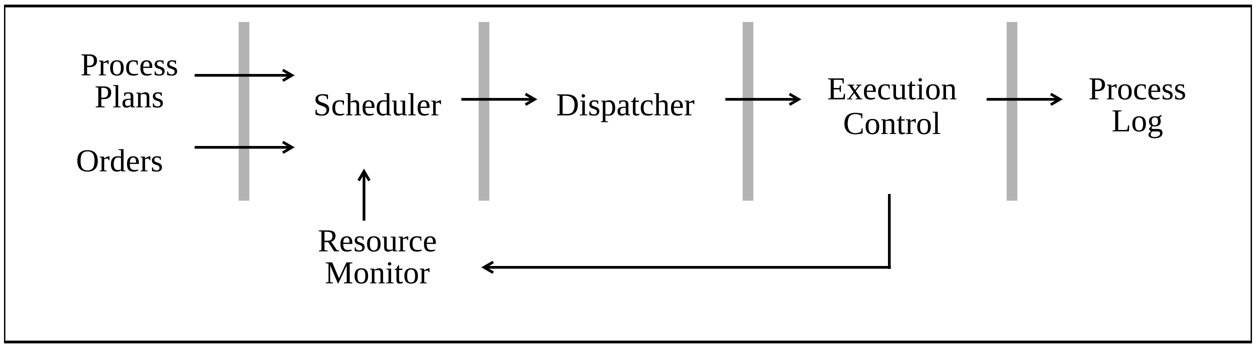

Figure 1

Data flow between components of a factory controller. Orders, process plans, and resource information communicate with scheduling. The Dispatcher resolves short term resource contention. The Execution control sequences operations and updates the process log and Resource Monitor. Thick vertical lines indicate ALPS interfaces.

A process specification language should communicate the processing information but not restrict downstream decision making. Process plans should allow the decoupling of various sub-tasks involved in manufacturing data preparation and production, e.g. specification of the task is decoupled from the scheduling policies applied, and resource management is decoupled from manufacturing control. Furthermore, each step in the preparation of manufacturing data (e.g. design, planning, production) must be decoupled from other steps.

Process specification languages (PSLs) have conflicting goals of simplicity, explicitness, and completeness of information. Simplicity is manifest in clarity, brevity, and the understandability of the language. Explicitness, however, requires all pertinent information to be communicated without hiding necessary information. Processing information should be explicit versus implicit in order for computers to properly understand it. A PSL must also provide for the complete specification of processing steps without omitting necessary information.

In order to satisfy the conflicting goals of simplicity, explicitness, and completeness, various means of abstraction should be supported, specifically hierarchical and macro abstraction. Hierarchical abstraction allows the clustering of processing steps into a single step at a higher control level. Macro abstraction allows for groups of processing steps to be clustered into a single step at the same control level. Thus, simplicity is maintained by limiting the explicit information at each level of control.

While the full specification of resources, tasks, and processing information should be possible, it should not be mandatory. An "intelligent" system may be able to calculate the necessary information while a more limited system may require the full specification. For example, an intelligent system could search the process plan for resource requirements, eliminating the need for explicit resource allocation nodes to be present in the plan.

For full specification of processing, a PSL must address the following aspects:

1) Processing precedence - determine the sequence of tasks. This is the basic capability of every PSL.

2) Alternative sequences - express different task sequences which provide the same result. Must also provide a means for a scheduler to determine which sequence is currently optimal. The decision should be deferred until scheduling or manufacturing time.

3) Parallel actions - explicitly show how multiple task sequences within a plan can be performed at the same time. It is assumed that separate plans can be executed in parallel as separate jobs.

4) Synchronization - provide for synchronization between multiple parallel task sequences in a plan (as in #3) and between multiple plans.

5) Resource monitoring - provide means for collecting and updating statistics for resource availability and utilization to support scheduling and resource allocation.

6) Post processing - provide a processing log to detail actual processing sequences used in manufacturing a part. A process specification language should have some means of capturing actual processing data to allow error tracing and to monitor performance. This aspect is critical for traceability, quality control and feedback control.

7) Extensibility - support extensibility by not constraining the user to a fixed functionality. Users must be able to customize process plans to support their facility.

Process specification languages must also support various degrees of implementation to allow incremental upgrading of systems. For example, a basic system can simply express the task precedence without alternative and parallel tasks. More sophisticated systems can express alternative and parallel tasks to allow for more productive run-time scheduling and selection of alternative tasks based on real-time information. Advanced implementations might involve resource management and monitoring of resources during processing to provide run-time statistics about machine utilization, tool wear, and inventory. This information can then be used to make better scheduling decisions.

Many languages exist to fulfill some of these requirements. However, in order to overcome perceived shortcomings with existing systems, a new process specification language was developed for use within the AMRF. This new language has been termed ALPS for "A Language for Process Specification".

III. Prior Work

Previous efforts in process specification provided a great deal of input to the ALPS project. Many languages are targeted for a particular application such as robotic assembly (Maimon 1986)(Adler 1986)(Homem 1986). These languages usually produce optimized solutions to a specific problem. The ALPS project aims to address a large application domain of manufacturing control and, while optimization is desirable in some cases, it is not an overriding concern.

Similarly, simulation languages (Taha 1988)(Pristker 1984)(Bobillier 1976) deal with the domain of process simulation which possesses many aspects of process specification but lacks the ability to carry over to a generalized control environment. Many of the ALPS split node variations and attributes are derived from simulation languages.

Many PSLs use directed graphs or AND/OR graphs as a means for communicating information (Homem 1986)(Passler 1982). AND/OR graphs provide the basis for the ALPS directed graphs but are extended to provide iteration, richer branching behavior, as well as enumerating different node classes.

Many PSLs (Rembold 1986)(Costa 1984) are based on existing programming languages like LISP, Pascal, or Ada. The difficultly arises because process engineers are generally not computer scientists, nor should they be required to be. This has spawned efforts to present the language with graphical or template-based front-end editors to facilitate program entry. The specification of the ALPS language uses a graphical front-end to an identical conceptual schema. There is no need for the "background" language because the front-end language gives full power.

ALPS provides a general vehicle for the implementation of a particular language. It is not intended to be an implementation language specification, but rather a conceptual facilitator of process specification. Although a language syntax has been developed and implemented as part of the research, the implemented language is not the primary thrust of the project.

IV. ALPS Structure

Directed Graph Notation

The primary purpose of a process specification language is to specify temporal relationships between process tasks. The ALPS language is based upon a directed graph notation to indicate the temporal relationship between nodes. The directed graph in Figure 2 shows that node A precedes

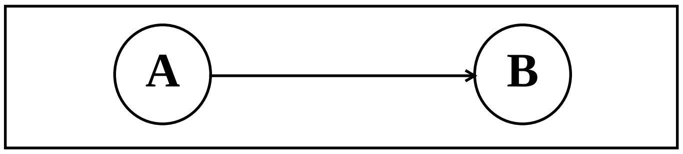

Figure 2
Simple directed graph indicating precedence relationship between node A and B.

node B in the graph and indicates that task A must complete prior to beginning task B. Many efforts in production control systems and simulation systems use directed graphs as their basis because it provides the required attributes of expression: simplicity, clarity, basic precedence, alternative sequences, parallelism, and abstraction.

Several benefits are obtained by using the directed graph notation: 1) the process plan is easily displayed for graphical editors and status-tracing tools, 2) parallel processing is explicitly shown, and 3) graphs are easily traversed by computer controllers. Sequencing processing tasks consists of connecting nodes in order. Parallel and alternative sequences are shown by nodes which have multiple successors. An example of a directed graph representation is shown in Figure 3.

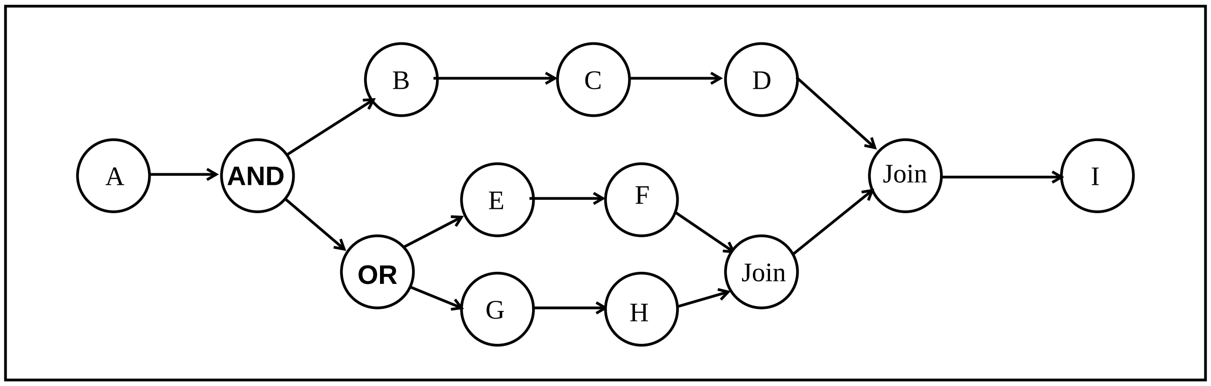

Figure 3

An example of the directed graph notation used for the process planning specification. Note the parallel activity (shown by the AND node) and the alternative paths (shown by the OR node). Nodes A-I indicate processing tasks to be performed. It should also be noted that each node in the figure has many attributes not shown here.

The following definitions are relevant.

Node -- a type of entity located at every vertex of a directed graph.

Arc -- a connector between two nodes denoting a temporal precedence relationship between nodes.

Path -- a sequence of one or more consecutive nodes and arcs.

Execution Path -- actual path selected for execution (when alternatives exist).

Task -- an elementary piece of work decomposed from a job.

Job -- one or more sequences of tasks to produce a change of state.

Target Controller -- control system which will execute a task.

Subordinate Controller -- control system at the next lower level in the control hierarchy. A controller decomposes jobs into simpler jobs which are given to subordinate controllers.

Supervisor Controller -- control system at the next higher level in the control hierarchy. Jobs are received from the supervisor controller.

Subordinate Plan -- a process plan at a subordinate level.

Parallel Tasks -- a group of tasks that may be done at the same time such as a job in a multi-tasking operating system.

Control hierarchy -- An arbitrary number of control levels that can be used to decompose work into more and more specific jobs (e.g. highest level controller might be a factory controller, lowest level controller might be an equipment controller). Higher levels of controllers have increasing abstraction and decreasing specialization.

Node Classes

The ALPS specification is built around a directed graph structure and is used to define the processing sequencing and specifications. There are seven major classes of graph nodes: termination, task, split, join, synchronization, resource, and information. The general structure of the nodes includes the following system defined attributes: 1) node identifier -- a required unique (for this plan) number to identify the node, 2) node name -- an optional attribute for a node commentary string, 3) node type -- a required attribute specifying the node class, 4) previous nodes (if applicable) -- required attribute containing the predecessor nodes in the graph, and 5) next nodes (if applicable) -- required attribute containing the successor nodes in the graph. Most nodes have other attributes specific to the node class. Full definitions and explanations of the nodes and node classes are outlined in the following sections.

Termination Nodes.

The START and END termination nodes delimit the beginning and end of the graph structure respectively. The START node is a unique node indicating the starting point of processing for this plan. The END node is a unique node and marks the last node in the graph. The termination nodes are place-holder nodes and do not contain processing information.

Task Nodes.

Two sub-classes of task node are defined: primitive tasks and decomposable tasks. A PRIMITIVE task node specifies activities which are directly executable by the target controller. For example, executing an NC code program might be a PRIMITIVE node for an equipment controller. Most PRIMITIVE task node attributes are user-defined for the particular task. Typical attributes would include tool identifier, speed, feed rate, resource utilization information, etc.

A decomposable task node consists of either a COMPLEX or MACRO task node. COMPLEX decomposable task nodes specify a process plan for decomposition and execution by a subordinate controller. This is the hierarchical abstraction mentioned in

Section IV. The specified plan is then read in and executed by the subordinate system. A MACRO decomposable task node specifies another process plan (ALPS file) as the collection of nodes to be processed in the place of this node. Such plans can themselves contain MACRO nodes. A MACRO decomposable task node specifies a process plan which is further decomposed by the same target controller. MACROs are analogous to subroutines which cluster multiple related statements into a single abstract statement. This is the macro abstraction mentioned in Section IV. Typical attributes for MACRO and COMPLEX task nodes are parameters for the specified plans. These attributes are user-defined.

Split Nodes.

Split nodes (branching nodes) allow the specification of different processing paths. Alternative paths, concurrent paths, and iterative paths all use the split node format. The two subclasses of split nodes are: PREDICATED and PARAMETERIZED. The split subclasses are a super-set of "AND" and "OR" nodes of generalized and/or graphs.

PREDICATED split nodes allow the selection of one or more paths based on Boolean predicate functions associated with each outgoing path. All predicates which evaluate to "true" (at run-time) are eligible to have their path followed. An M-number (named after the SLAM M-number (Pristker 1984)) is specified to determine the maximum number of paths which may be taken from the node. There is a practical minimum of one path taken. The ELSE clause is taken in case no predicates evaluate to true. A predicated split node could be used to select between alternative paths based on some simple predicate (e.g. surface_finish > x).

PARAMETERIZED split nodes allow the selection of one or more paths by specifying parameters associated with each outgoing path. A local scheduler uses the parameters and values to decide which path(s) to take. An M-number is specified to determine the actual number of paths to take (e.g. take the three best paths). It should be noted that the M-number has a subtle difference in the parameterized and the predicated split nodes. This is due to the nature of the nodes and should not be viewed as an inconsistency. A parameterized split node could be used to choose dynamically between alternative paths based on run time information and local scheduling policies. Typical attributes might include cost, duration, and resource requirements.

All split node paths provide an optional local function attribute which is evaluated after a path is selected. These functions can manipulate local data and may affect future path evaluations. Path evaluations, by convention, proceed from first-specified to last-specified.

For both the PREDICATED and PARAMETERIZED split node, a timing attribute is specified to indicate whether the appropriate paths should be executed in series or in parallel. Serial execution of paths provides for a convenient shorthand to enumerate multiple processing sequences which may be done in any order but may not be done in parallel. Parallel execution provides explicit parallelism in the process plan.

<u>Join Nodes</u>

Join nodes are required to bring multiple paths back together after a split node. Each split node has a corresponding join node and split-join pairs may be nested to an arbitrary depth. A join node also specifies which split node is paired with it.

There are two join subclasses indicating the two ways paths can be taken from a split node. Both join subclasses join multiple paths. A MULTIPLE join node joins paths where more than one path was executed, while a SINGLE join node joins paths only one of which was actually executed. These subclasses provide duplicate information to the controller. Later versions of ALPS may drop this subclass specification requirement, but initially it was felt that the join nodes should explicitly (and redundantly) state their class.

The SINGLE join subclass has two node subclasses: ALTERNATIVE and ITERATION. ALTERNATIVE join nodes join paths taken from a split which selected one path from several alternatives. The ITERATION join provides a special join for developing looping constructs. The ITERATION join node is the only join node which precedes the corresponding split.

<u>Resource Nodes</u>

Resource nodes provide for explicit specification of resource management. The subclasses of resource nodes are: ALLOCATE and DEALLOCATE. ALLOCATE and DEALLOCATE are complementary actions.

ALLOCATE and DEALLOCATE nodes provide a resource-locking mechanism to prevent preemption of resources at critical times. An ALLOCATE node marks the start of this critical section in which no allocated resources may be given up. To prevent deadlock, an ALLOCATE node must specify all non-sharable resources required for the entire critical section, and an allocation succeeds only if all resources are successfully allocated. A DEALLOCATE node may specify one or more allocated resources, and multiple deallocations may be required for a single allocate.

The existence of the ALLOCATE/DEALLOCATE nodes defines critical sections of resource usage. Resource preemption is allowed between any two nodes <u>not</u> contained in a critical section for that resource. This allows a controller to preempt resources in favor of higher priority jobs and tasks.

The ALLOCATE node indicates additional information relating to the time and duration of resource usage. This information is provided as an estimation of times for a semi-intelligent controller. Fully intelligent controllers could obtain the same information by searching ahead in the graph.

Synchronization Nodes

Synchronization nodes provide several means for coordinating multiple parallel tasks. The background for this class of node is presented in (Hoare 1985)(Dijkstra 1968). There are two basic categories of synchronization node: semaphore and clock. The semaphore synchronization nodes are divided into subclasses of: RENDEZVOUS, SIGNAL, AWAIT, LOCK, and UNLOCK. The clock-based synchronization nodes consist of the subclasses: WAIT and DELAY.

SIGNAL and AWAIT synchronization nodes provide the basic means to coordinate several parallel tasks. A SIGNAL node indicates that a specific event (semaphore) has occurred. An AWAIT node requires execution to halt along this path at this node until the specified event (semaphore) has been signalled. SIGNAL/AWAIT node coordination may be between multiple paths in a single plan or between multiple paths in different plans. Furthermore, many paths may be awaiting a single signal. A SIGNAL node defines the signalled event by uniquely naming a semaphore. The full semaphore name consists of the plan identifier of this SIGNAL node, the plan version number, and a unique string name. The combination of these three ingredients provides the complete semaphore name. AWAIT nodes specify the complete semaphore name to allow synchronization between plans.

A RENDEZVOUS node coordinates a synchronized data transfer between two paths. Although the SIGNAL/AWAIT nodes are sufficient to implement a rendezvous, a separate node provides increased clarity and understanding. Upon reaching a RENDEZVOUS node, execution along that path must stop until the corresponding RENDEZVOUS node is reached and the data transfer completed.

The LOCK and UNLOCK provide a general purpose resource counter type of semaphore. (LOCK is equivalent to Dijkstra’s P operation and UNLOCK is equivalent to the V operation.) The mechanism is provided for general purpose use and may not be necessary in all applications.

Timing synchronization is provided by two clock synchronization nodes: WAIT and DELAY. The WAIT synchronization node will pause execution until a specified time arrives. This is useful for starting tasks no sooner than a specified time. The DELAY node allows processing to pause for no less than the specified amount of time.

Information Nodes

INFORMATION nodes provide for general purpose, user-definable operations such as database queries, parameter bindings, computations, or other operations not covered specifically by other nodes. The format of the user-defined expressions is application specific; for the initial ALPS implementation a generalized LISP expression syntax was used.

Global Plan Information

In addition to the directed graph nodes, there is information in a process plan which pertains to the entire plan. This includes the following required attributes: plan identifier, plan version number, and target controller. The combination of plan identifier and plan version number uniquely identifies the plan, while the target controller indicates which controller should be executing this particular plan. Optional and user-defined attributes which might also be included as global plan information are: process engineer, plan status, authorization code, plan parameters, local constant definitions, local variables, revision control information, etc.

Conceptual Schema

A powerful method of supporting the ALPS language when constructing process plans is to use a database. The act of defining and attributing nodes of different classes consists of simply populating this database. The database itself can assure the adherence to consistency constraints such as precedence rules between nodes, ensuring correct data types for attributes, and referential uniqueness. Object-oriented databases make this type of function particularly easy. In addition, creation of a physical database from a conceptual schema is easier with an object-oriented tool than it is with a relational database.

Our work on database support for ALPS began with the definition of the conceptual ALPS schema of the node behavior for the ALPS language. This was done using the NIAM methodology, with the working schema shown in the Appendix. This formal schema concisely defines the behavior of the node classes described above. Physical databases on a variety of platforms can be unambiguously derived from the NIAM "blueprint". It is important to note that this schema does not address the information used during the problem-solving stage of process planning - that is being defined in the Plan Formulation schema (see Ray 1989). Rather, the ALPS schema describes the behavior of the nodes in the ALPS precedence graph, plus the global plan information contained in the plan header. Thus, the ALPS schema is solely concerned with describing the completed process plans and not with the information used to produce the plan.

Once the ALPS schema was defined, an object-oriented database was implemented. Classes were defined by converting each non-lexical object in the NIAM diagram (solid circles) to a database class definition. The object-oriented database systems used were the Statice* system, written by Symbolics, and Gbase, by Graphael. The underlying Lisp language implementation makes the support of object-oriented databases particularly easy, since the language itself already possesses most of the necessary behavior.

With the database in place, any manufacturing system is able to access the plan information by means of standard database queries. This database approach forms part of a larger architectural design addressing the integration of manufacturing data preparation functions.

V. Conclusions

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

The ALPS activity supports work being done in the AMRF and in the development of the PDES 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 resource schema, and the Plan Formulation schema.

Future work will focus on refining and testing ALPS. Several aspects of ALPS are either missing or not thoroughly defined, such as deadline management, parameter passing and necessarily simultaneous processes. The continuing work on ALPS will address these and other unresolved issues.

A physical file format which supports the ALPS schema has also been developed and is being used in a prototype integrated manufacturing environment. A generic manufacturing controller is being developed to accept ALPS data to control the manufacturing environment. Details of how ALPS is being used to drive a controller can be found in (Newton 1990.)

* Certain commercial equipment, instruments, or materials are identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

VI. References

(Adler 1986) Adler, A., "TDL A Task Description Language for Programming Automated Robotic Workcells," *IEEE International Conference on Systems, Man, and Cybernetics*, 65-68, Oct 14-17, 1986.

(ANSI 1981) American National Standards Institute, "Digital Representation for Communication of Product Definition Data," *American National Standard ANSI Y14.26M-1981,* American Society of Mechanical Engineers, New York, 1981.

(Bobillier 1976) Bobillier, P.A., Kahan, B.C., and A.R. Probst, *Simulation with GPSS and GPSS V,* Prentice-Hall, Englewood Cliffs, NJ,1976.

(Costa 1984) Costa, A. and M. Garetti, "Design of a Control System for a Flexible Manufacturing Cell," *Journal of Manufacturing Systems*, Vol. 4, No. 1, 1984.

(Dijkstra 1968) Dijkstra, E.W., "Cooperating Sequential Processes," in *Programming Languages*, ed. F. Genuys, Academic Press, New York, NY, 1968.

(Furlani 1983) Furlani, C., et al., "The Automated Manufacturing Research Facility of the National Bureau of Standards," *Proc. of the Summer Simulation Conference,* Vancouver, BC, Canada, July 11-13, 1983.

(Henghold 1989) Henghold, William M., Gerald C. Shumaker and Leonard Baker, "Considerations for the Development and Implementation of PDES within a Government Environment," *Report AFWAL-TR-89-8009,* Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Ohio, February, 1989.

(Hoare 1985) Hoare, C.A.R., *Communicating Sequential Processes,* Prentice-Hall, Englewood Cliffs, NJ, 1985.

(Hocken 1983) Hocken, R. and P. Nanzetta, "Research in Automated Manufacturing at NBS," *Manufacturing Engineering*, 91, Vol. #4, 1983.

(Homem 1986) Homem de Mello, L.S. and A.C. Sanderson. "AND/OR Graph Representation of Assembly Plans," *Proc. of AAAI-86,* Vol. 2, 1113-1119, Philadelphia, PA, Aug 11-15, 1986.

(Maimon 1986) Maimon, O.Z., "A Generic Multirobot Control Experimental System," *Journal of Robotic Systems*, 3(4), 451-466, 1986.

(McLean 1983) McLean, C.R., Mitchell, M., and E. Barkemeyer, "A Computing Architecture for Small Batch Manufacturing," *IEEE Spectrum*, May 1983.

(McLean 1985) McLean, C.R., "An Architecture for Intelligent Manufacturing Control," *Proc. Summer 1985 ASME Conference,* Boston, MA, August 1985.

(Nanzetta 1984) Nanzetta, P., "Update: NBS Research Facility Addresses Problems in Setups for Small Batch Manufacturing," *Industrial Engineering*, June 1984.

(Newton 1990) Newton, E.C. and B.A. Catron, "Control Architectures for Manufacturing Data Preparation," to be published as an NIST Interagency Report.

(Passler 1982) Passler, E. et al., "Production System Design: A Directed Graph Approach," *Journal of Manufacturing Systems,* Vol. 2, No. 2, 107-116.

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

(Ray 1989) Ray, S.R. "A Modular Process Planning System Architecture," *Proc. of 1989 IIE Integrated Systems Conference & Society for Integrated Manufacturing Conference*, Atlanta, GA, November 1989.

(Rembold 1986) Rembold, U. and W. Epple, "Present State and Future Trends in the Development of Programming Languages for Manufacturing," *Computer-Aided Design and Manufacturing-Methods and Tools*, 1986.

(Simpson 1982) Simpson, J.A., Hocken, R.J., and J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards," *Journal of Manufacturing Engineering,* 1, Vol. #1, 1982.

(Taha 1988) Taha, H.A., " The SIMNET Simulation Language," *Computers ind. Engng*, Vol. 14. No. 3, 281-295, 1988.

VII. Appendix

The following statements describe the entities which appear in the diagram shown in Figure 4 at the end of this Appendix. Each word written in uppercase letters corresponds to an entity in Figure 4. The statements, along with the listed constraints and the diagram, constitute the formal schema definition for ALPS. It should be noted that the schema as presented is still incomplete. All of the node subclasses without associated relations will eventually be defined with attributes specific to each subclass. For now, all such attributes are handled by the generic entity ATTRIBUTE, which is inherited by all nodes. Statements are numbered for ease of referencing.

01) A PLAN is made of one or more NODES.

02) A PLAN has a unique PLAN ID.

03) A PLAN has a single HIERARCHY LEVEL.

04) A PLAN has a single VERSION.

05) A PLAN has a single TARGET.

06) A PLAN is referred to by zero, one or many DECOMPOSABLE TASK NODES.

07) A NODE has a single NODE NUMBER.

08) A NODE has a single NODE TYPE.

09) A NODE has a single NODE NAME.

010) A NODE belongs to a single PLAN.

011) A NODE is uniquely defined by its NODE NUMBER and the PLAN it belongs to.

012) A NODE contains zero, one or many ATTRIBUTES.

013) An ATTRIBUTE qualifies a single NODE.

014) An ATTRIBUTE has a single ATTRIBUTE NAME.

015) An ATTRIBUTE has a single ATTRIBUTE VALUE.

016) A NODE is either a SINGLE PREDECESSOR NODE, a SINGLE SUCCESSOR NODE, or both.

017) A SINGLE PREDECESSOR NODE is preceded by zero or one NODES.

018) A SINGLE SUCCESSOR NODE is succeeded by zero or one NODES.

019) A SINGLE PREDECESSOR NODE is either a SPLIT NODE or a NON-BRANCHING NODE, but not both.

020) A SPLIT NODE is succeeded by more than one SINGLE PREDECESSOR NODE.

021) A SINGLE SUCCESSOR NODE is either a JOIN NODE or a NON-BRANCHING NODE, but not both.

022) A JOIN NODE is preceded by more than one SINGLE SUCCESSOR NODE.

023) A NON-BRANCHING NODE is a subclass of SINGLE SUCCESSOR NODE and a subclass of SINGLE PREDECESSOR NODE.

024) A SPLIT NODE is either a PARAMETERIZED SPLIT NODE or a PREDICATED SPLIT NODE, but not both.

025) A JOIN NODE is either a MULTIPLE JOIN NODE or a SINGLE JOIN NODE, but not both.

026) A SINGLE JOIN NODE is either an ITERATION SINGLE JOIN NODE or an ALTERNATIVE SINGLE JOIN NODE, but not both.

027) A NON-BRANCHING NODE is either a RESOURCE NODE, TERMINATION NODE, TASK NODE, INFORMATION NODE or a SYNCHRONIZATION NODE, but not more than one of them.

028) A RESOURCE NODE is either an ALLOCATE RESOURCE NODE or a DEALLOCATE RESOURCE NODE, but not both.

029) A TERMINATION NODE is either a START TERMINATION NODE or an END TERMINATION NODE, but not both.

030) A SYNCHRONIZATION NODE is either a RENDEZVOUS, LOCK, UNLOCK, SIGNAL, AWAIT, DELAY or TIMED WAIT SYNCHRONIZATION NODE, but not more than one of them.

031) A TASK NODE is either a DECOMPOSABLE TASK NODE or a PRIMITIVE TASK NODE, but not both.

032) A DECOMPOSABLE TASK NODE is either a COMPLEX DECOMPOSABLE or MACRO DECOMPOSABLE TASK NODE, but not both.

033) A DECOMPOSABLE TASK NODE refers to a single PLAN ID.

034) A DECOMPOSABLE TASK NODE refers to zero or one plan VERSION.

Notes and Constraints:

01) Only the leaves of the node taxonomy are ever instantiated. In other words, the only classes to contain actual data will be those which have no subclasses. For example, one would never have an instance of a NON-BRANCHING NODE, but one might have an instance of a START TERMINATION NODE, which inherits properties from NON-BRANCHING NODE.

02) The inverse relations of "succeeded by" and "preceded by" are never used in the implementation, as this would cause ambiguity in the schema.

03) Every SPLIT NODE must have a corresponding JOIN NODE belonging to the same PLAN.

04) Every ITERATION SINGLE JOIN NODE must occur earlier in a PLAN than its corresponding SPLIT NODE. That is, the ITERATION SINGLE JOIN NODE must be a predecessor (but not an immediate predecessor) of its corresponding SPLIT NODE.

05) Every other subclass of JOIN NODE must occur later in a PLAN than its corresponding SPLIT NODE.

06) A JOIN NODE must correspond to the nearest predecessor SPLIT NODE belonging to the same PLAN. That is, strict recursion is enforced - a JOIN NODE "closes up" the latest SPLIT NODE, not some earlier one.

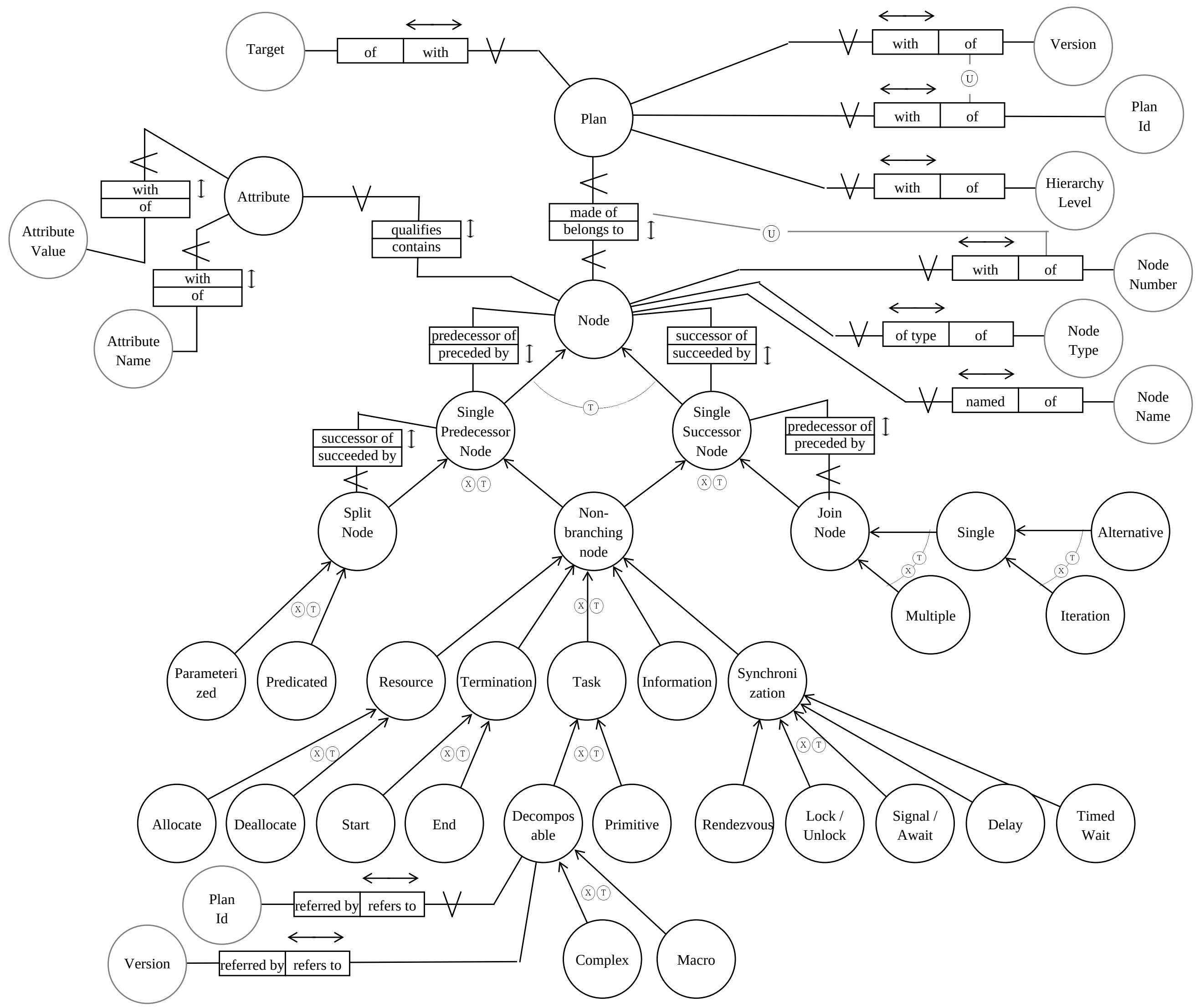

Figure 4

NIAM diagram of the ALPS schema. This diagram, together with the statements and constraints appearing in this Appendix constitute the complete ALPS schema definition.