NBSIR 88-3770

AN AMPLE VERSION 0.1 PROTOTYPE: THE HWS IMPLEMENTATION

April 21, 1988

By:
Herbert T. Bandy Victor E. Carew, Jr. Jack C. Boudreaux

U.S. DEPARTMENT OF COMMERCE National Bureau of Standards Gaithersburg, Maryland

An AMPLE Version 0.1 Prototype: The HWS Implementation

H.T. Bandy V.E. Carew, Jr. J.C. Boudreaux

27 April 1988

CONTENTS

FIGURES

An AMPLE Version 0.1 Prototype: The HWS Implementation

I. INTRODUCTION

The Automated Manufacturing Programming Language Environment (*AMPLE*) system functioned as the means for programming the Horizontal Workstation (HWS) during the March 1987 benchmark test of the Automated Manufacturing Research Facility (AMRF). The AMRF has been established at the National Bureau of Standards for research concerning control interfaces between automated components of flexible manufacturing systems, and for examining related issues of standards and measurement [18]. HWS, the AMRF workstation for horizontal milling, has served as a laboratory not only for the development of several prototype controllers, interfaces and devices, but also for experimental implementation and demonstration of the *AMPLE* system.

AMPLE is a general system for the definition and verification of manufacturing control data [5] and [6]. Built around a central kernel called *AMPLE/core, AMPLE* provides an integrated system of software tools for translating product design and process planning specifications into equipment-level control programs.

In this report the modules of the implementation of the *AMPLE* Version 0.1 prototype for HWS will be described. All of the *AMPLE* software used to support HWS runs on a Silicon Graphics IRIS model 3020 workstation. This workstation features a Motorola MC68020 CPU running UNIX System V with Berkeley 4.2 enhancements. Enhanced graphics capability is available through a custom VLSI chip set that permits real-time animation of complex polygonal models. This report will discuss the following modules:

- **AMPLE Process Planning Interface (APPI).**
  The purpose of this module is to determine that process plans are correct and complete before they are made available to HWS.[1] A process plan is correct if it conforms to the syntax of the AMRF flat file format, and if the work elements in the procedure section of the plan can in fact be performed by HWS. A process plan is complete if all work elements needed to manufacture the part have been provided in the proper sequence. By first parsing and then analyzing a process plan, APPI determines whether these criteria are satisfied.

- **AMPLE Real-Time Control Interface (ARTCI).**
  This module accepts high-level instructions, either interactively from the user interface or remotely from APPI, and then generates equipment-level control data, and data to drive the animation package.

---

[1] The *AMPLE* Real-Time Control Interface, in its interactive data preparation mode described in Section IV, was used to create the process plans used in the March 1987 AMRF benchmark test. The process plans were created and stored in advance, so that a command from HWS to *AMPLE* during the benchmark test would initiate APPI processing of any specified plan.

- **AMPLE Workstation Animation Package (AWAP).**
  This module accepts input from ARTCI and generates an animated preview of all of the physical motions that the associated control data, if actually executed, would produce. In this manner, AWAP allows off-line validation and testing of control programs.

As Appendix A, the *AMPLE* module which functions as an interface to HWS is also included:

- **AMPLE Communication Package (Acomm).**
  This module provides a communication port between HWS and *AMPLE*. Though several different kinds of data can be transmitted, the most important information is contained in process plans.

This report will describe the HWS implementation of the *AMPLE* prototype as it existed during the March 1987 AMRF public benchmark test. The significant enhancements in the system which have been developed since that time will be presented in separate reports.

1. AN ORIENTATION FOR THE READER

Appendices and a glossary are provided for clarification of system design considerations and of certain terminology. Without the need to consult other references, the reader of this document may expect to derive a basic understanding of how to design and generate programs to make parts in HWS in the AMRF. The document does not cover generation of NC data for the machine tool or grip and position data for the robot control system [21]; nor does it cover programming the robot vision system [14].

The described methods of process plan verification and control data generation are implementations of principles which are either well understood or documented elsewhere. On the other hand, since this document introduces AWAP as a new animation software design, the section containing pertinent explanations is more elaborate than other sections.

To present the reader with a concise view of the HWS implementation of *AMPLE*, this document will use a *pseudocode notational system* which may be loosely defined as structured English with liberal adaptations from the Ada programming language.[2] Specifically, the pseudocode will specify relationships between symbols by using Ada **type** constructors such as **array** and **record**, and will characterize the flow of control by **if-then-else**, **loop**, and **case**. Since the pseudocode presents an abstract and very condensed description of a large software system, the reader is advised to consult the surrounding narrative for more detailed explanations.

---

[2] Ada is a registered trademark of the U.S. Government, Ada Joint Program Office (AJPO). This language is defined in ANSI/MIL-STD 1815A-1983, *American National Standard Reference Manual for the Ada Programming Language.*

II. AN OVERVIEW OF THE AMRF HORIZONTAL WORKSTATION

HWS is one of six workstations in the AMRF. It features a customized Cincinnati Milacron T3 robot with an NBS-developed vision and control system, a White Sundstrand horizontal machining center, a Kardex materials buffering system, and several custom-built flexible fixtures.

AMPLE provides support for five controllers in HWS [16]. It provides three levels of control data for the Horizontal Workstation Controller [19], commands and data for the fixturing controller [17], and commands for the High Level Machine Tool Controller [11], the Real-time Control System of the robot [21], and the Materials Buffering Controller [9]. An introduction to each controller, as well as a brief example of the data provided to it by *AMPLE*, will now be presented. Note that in all of the examples presented in this section, the term *MacRow* refers to control code macros designed specifically for HWS.

1. HORIZONTAL WORKSTATION CONTROLLER (HWSC)

The Horizontal Workstation Controller is a finite state machine implementation of a hierarchical control system. It is implemented in the FORTH programming language, and runs on a Motorola MC68000-based computer system. The workstation breaks commands down to three hierarchial levels, and the *AMPLE* system provides all three levels of control data for the workstation.

AMPLE generates three levels of control data for HWSC. The first level of data describes in a very general way what type of operation the system is about to perform. The content and syntax of these data are shown in the following pseudocode.

```
type WORKSTATION-LEVEL-0-COMMAND-STRING is record
  operation : 10 bytes;
  occurrence : 2 bytes;
  count : 2 bytes;
  argument : 2 bytes
  MacRow-name : 10 bytes;
end record;
```

An example of these data is:

operation	occurrence	count	unknown	MacRow name
1234567890	12	12	12	1234567890
MAKE	1	1	1	L0_MAKE_IT

The second level of control data describes how the system will behave in more specific terms. These data include the order in which major operations are to occur, and which equipment MacRows will be used. Here is a pseudocode description of the string:

```
type WORKSTATION-LEVEL-1-COMMAND-STRING is record
  operation : 10 bytes;
  occurrence : 2 bytes;
  count : 2 bytes;
  argument : 2 bytes
  MacRow-name : 10 bytes;
end record;
```

An example of these data is shown below:

```
 operation  occurrence     count     unknown    MacRow name
1234567890      12          12         12       1234567890

LOAD             1           1          1       L1_LOAD_1
```

The third and final level of control data consists of the actual commands to be sent to each controller. At this level, there are two parts to every data set. The first part is the actual command name and instance number, and the second part is a literal ASCII representation of the command with the parameter values filled in. Here is a pseudocode expansion of the level 2 command string:

```
type WORKSTATION-LEVEL-2-COMMAND-STRING is record
  operation : 10 bytes;
  operation-occurrence : 2 bytes;
  device-name : 10 bytes;
  command : 10 bytes;
  command-occurrence : 10 bytes;
end record;
```

A sample of the actual command data for a part is shown below:

```
operation    occurrence  device name   command     occurrence
1234567890      12       1234567890  1234567890       12

LOAD             1       ROBOT       MOVE              1
```

The commands are described in the sections for each controller.

2. COMMAND LEVEL INTERFACE FOR RCS

The Real-time Control System (RCS) for the robot is also implemented as a finite state machine [1]. It too is implemented in the programming language FORTH, but uses multiple

Intel 8086 CPUs to perform computations. The Real-time Control System also integrates with a vision system and a safety system. *AMPLE* provides HWS with commands to be issued to RCS that include source and destination locations and grip numbers. All other information for the robot is entered through the user interface of RCS.

The command strings for the robot control system are ASCII strings that are sent from the Workstation Controller to RCS. These strings include a command to be executed, an object name, an object serial number, a location where the object may be found, a destination location where the object is to be placed, and the location where the robot is to go when it has completed the move. Some of these arguments are optional. The flag that appears at the start of the string tells the workstation controller which of the fields in the command string are to be filled with the current system values. The other integer numbers in the string identify grip poses and tray sectors to be used for the move. Pseudocode describing this string is shown below:

```
type ROBOT-COMMAND-STRING is record
  robot-string-flag : 2 bytes;
  robot-command : 16 bytes;
  serial-number : 16 bytes;
  source-grip-number : 2 bytes;
  source-location : 16 bytes;
  source-sector : 1 byte;
  destination-grip-number : 2 bytes;
  destination-location : 16 bytes;
  destination-sector : 1 byte;
  end-location : 16 bytes;
  end-sector-number : 1 byte;
end record;
```

An example of these data is shown below:

```
flag        command          object name        serial number
 12    1234567890123456  1234567890123456   1234567890123456

 00    MOVE              FL209-B            J13472


flag   source location  flag    flag   destination location
 12    1234567890123456  1       12      1234567890123456

 +1    BUFFER            1       +2      VFIXTURE
```

```
flag     end-at location      flag
 1       1234567890123456      1

 1       SAFE                  0
```

For a detailed description of these parameters, and a description of other robot programming operations, see [21].

3. HIGH LEVEL MACHINE TOOL CONTROLLER (HLMC)

The High Level Machine tool Controller is used to enhance the capabilities of the existing machine tool controller. The HLMC allows programs to be transferred to and from the machine tool, allows remote monitoring of the machine's status, and permits commands to be issued to the machine tool from a remote source. It also provides sensory capability that the machine would otherwise lack. The High Level Machine tool Controller is implemented in FORTH on an Intel 8086 CPU. *AMPLE* provides ASCII commands for the Workstation Controller to issue to HLMC.

The command data that *AMPLE* generates for HWSC to send to HLMC include a flag, a command, and an object. The flag field is used by the workstation controller to indicate which, if any, fields are to be filled in with current system values. The command field contains the command to be executed by the HLMC, and the object field describes the object to be acted upon by the command. A pseudocode definition of the HLMC command is shown below:

```
type NC-COMMAND-STRING is record
  nc-string-flag : 2 bytes;
  nc-command : 30 bytes;
  nc-argument : 20 bytes;
end record;
```

Here is a sample command for HLMC:

```
flag                  command                        object
          0        1         2         3  0        1         2
 12       12345678901234567890123456789  12345678901234567890

 00       MACHINE                        FL209-B
```

4. HORIZONTAL WORKSTATION FIXTURING CONTROLLER (HWFC)

The Horizontal Workstation Fixturing Controller addresses both primary fixtures in HWS. One fixture holds prismatic parts and the other holds turned parts. Both were designed at NBS

and can be manipulated under computer control to hold a wide variety of parts. The fixturing controller is implemented in FORTH on a Motorola MC68000-based system.

AMPLE generates two types of data for the fixturing controller -- command data and fixture data. Command data identify particular sets of fixture data, and are to be sent by HWSC to the fixturing controller when fixture data are to be executed. Fixture data are to be stored in the fixturing controller in advance of the issuing of commands, and denote the action that the fixturing controller will take when it receives a command.

```
type FIXTURE-COMMAND-STRING is record
  fixture-string-flag : 2 bytes;
  fixture-command : 10 bytes;
  part-number : 10 bytes;
  fixture-name : 10 bytes;
end record;
```

An example of HWFC data is shown below:

```
 flag       command       part name     fixture name
12         1234567890     1234567890      1234567890

00         LOCK           FL209-B         PFIXTURE
```

Information about the fixture data may be found in [17].

5. MATERIALS BUFFERING CONTROLLER (MBC)

The Materials Buffering Controller was built using the same set of tools that were used to build the Workstation Controller. *AMPLE* generates commands for the Workstation Controller to send to the MBC, and the workstation fills in some information during execution based on the current state of the system. The Materials Buffering Controller is implemented in FORTH on a Motorola MC68000-based system.

The command data for the Materials Buffering Controller are generated by *AMPLE*, and sent by the HWSC as commands to the MBC. These data may be represented as follows:

```
type MBC-COMMAND-STRING is record
  mbc-string-flag : 2 bytes;
  mbc-command : 16 bytes;
  object-type : 16 bytes;
  object-name : 16 bytes;
  serial-number : 16 bytes;
end record;
```

The flag indicates to the workstation which fields in the command string should be filled with current system values. The command string describes the action that the materials buffer is to take. The object type field characterizes the object as blank, finished, a tool, or some other item. The serial number field is usually filled in at execution time by the workstation controller. A sample of the MBC data is shown below:

```
 flag          command             object type          object name
12         1234567890123456    1234567890123456    1234567890123456

00         PRESENT             PART                FL209-B

 serial number
1234567890123456

13897
```

III. AMPLE PROCESS PLANNING INTERFACE (APPI)

A process plan is a report that describes the high-level workstation operations which define a manufacturing process. Since such reports are used by systems throughout the AMRF, a syntax for a *flat file format* for process plans has been defined. A full account of the features of AMRF process plans is presented in [8]. An example is shown in Figure 1. For the purposes of this document, only a brief list of the major features of this plan need be discussed.

Process plans in flat file format are text files, which means that every process plan must be resolved into *tokens*. Some tokens, such as reserved keywords and punctuation symbols, have special syntactic roles; others, such as strings or user-defined keywords, have only system-dependent significance.

The overall structure of plans may be described in terms of certain reserved keywords. First, as Figure 1 shows, the first token and the last token of process plans must be keywords:

```
--PROCESS_PLAN--

   the process plan text

--END_PROCESS_PLAN--
```

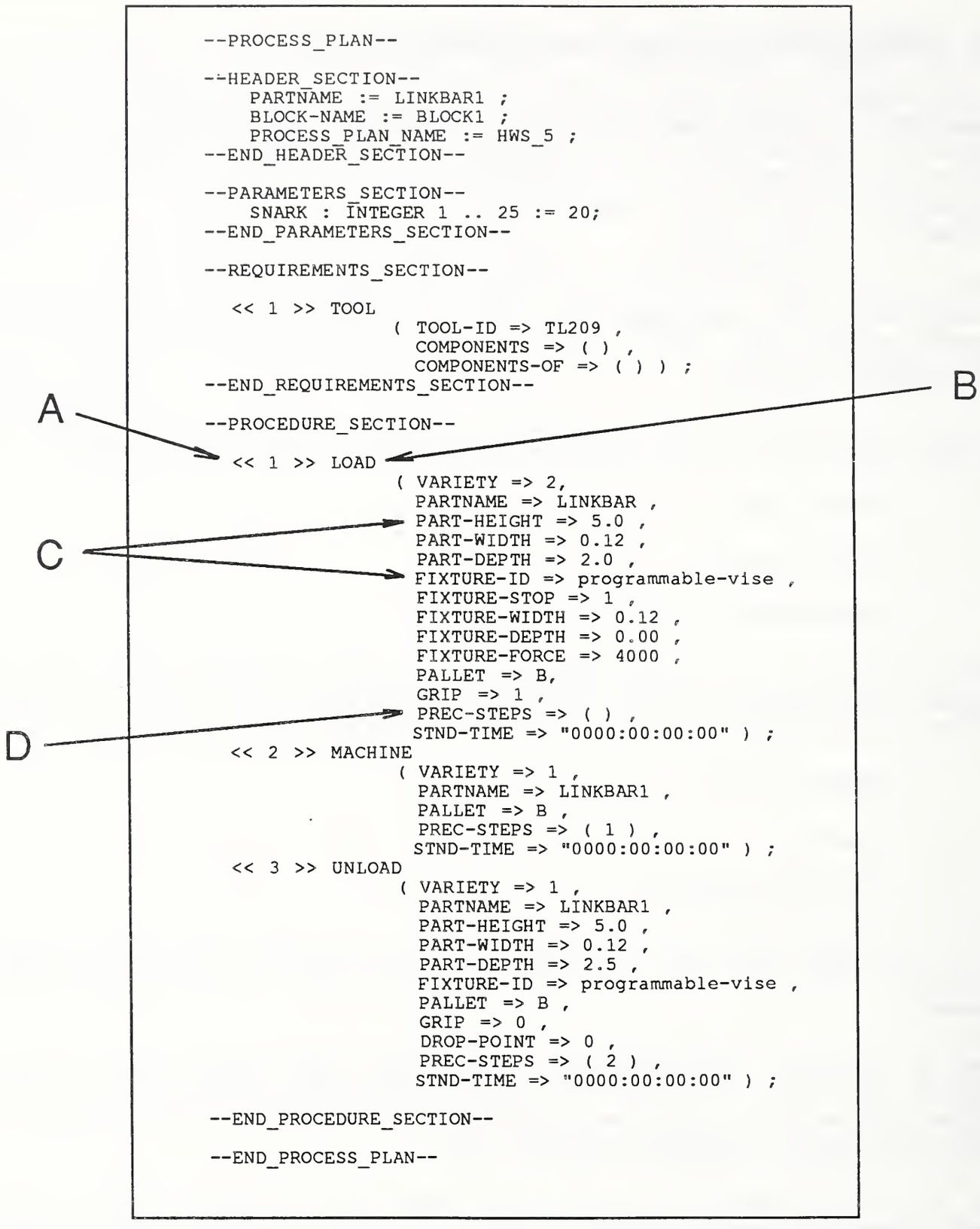
The process plan text is further divided into four sections: the *header* section, the *parameter* section, the *requirement* section and the *procedure* section. Each of these sections is introduced and terminated by keywords. For example, the header section has the form:

```
--HEADER_SECTION--

   header lines

--END_HEADER_SECTION--
```

where each header line consists of a header element name and the value which is assigned to it. Notice that in this section and in all other sections lines are terminated by semicolon punctuation.

From the programmer's perspective, header element names and admissible ranges of values are relatively unrestricted. In fact, the only header element name which must be present in all process plans for HWS is the keyword PARTNAME whose assigned value will be used as the basic name of the part throughout the *AMPLE* system.

The only strictly enforced rule governing the header section is that of *component alignment*. Header lines must have the following tokens in the order specified: a legal process plan keyword, the assignment symbol, a legal value, and finally the semicolon line terminator. (See Appendix B.) Any deviation of these alignment rules is a fatal error.




```
--PROCESS_PLAN--

--HEADER_SECTION--
    PARTNAME := LINKBAR1 ;
    BLOCK-NAME := BLOCK1 ;
    PROCESS_PLAN_NAME := HWS_5 ;
--END_HEADER_SECTION--

--PARAMETERS_SECTION--
    SNARK : INTEGER 1 .. 25 := 20;
--END_PARAMETERS_SECTION--

--REQUIREMENTS_SECTION--

  << 1 >> TOOL
               ( TOOL-ID => TL209 ,
                 COMPONENTS => ( ) ,
                 COMPONENTS-OF => ( ) ) ;
--END_REQUIREMENTS_SECTION--

--PROCEDURE_SECTION--

  << 1 >> LOAD
               ( VARIETY => 2,
                 PARTNAME => LINKBAR ,
                 PART-HEIGHT => 5.0 ,
                 PART-WIDTH => 0.12 ,
                 PART-DEPTH => 2.0 ,
                 FIXTURE-ID => programmable-vise ,
                 FIXTURE-STOP => 1 ,
                 FIXTURE-WIDTH => 0.12 ,
                 FIXTURE-DEPTH => 0.00 ,
                 FIXTURE-FORCE => 4000 ,
                 PALLET => B,
                 GRIP => 1 ,
                 PREC-STEPS => ( ) ,
                 STND-TIME => "0000:00:00:00" ) ;
  << 2 >> MACHINE
               ( VARIETY => 1 ,
                 PARTNAME => LINKBAR1 ,
                 PALLET => B ,
                 PREC-STEPS => ( 1 ) ,
                 STND-TIME => "0000:00:00:00" ) ;
  << 3 >> UNLOAD
               ( VARIETY => 1 ,
                 PARTNAME => LINKBAR1 ,
                 PART-HEIGHT => 5.0 ,
                 PART-WIDTH => 0.12 ,
                 PART-DEPTH => 2.5 ,
                 FIXTURE-ID => programmable-vise ,
                 PALLET => B ,
                 GRIP => 0 ,
                 DROP-POINT => 0 ,
                 PREC-STEPS => ( 2 ) ,
                 STND-TIME => "0000:00:00:00" ) ;

--END_PROCEDURE_SECTION--

--END_PROCESS_PLAN--
```

Figure 1. Example Process Plan

This text is an example of a process plan in flat file format: (A) the step number for the first step of the process plan; (B) the work element name of the first step; (C) an attribute-value pair; and (D) the PREC-STEPS attribute which indicates the precedence of steps.

APPI is primarily concerned with the *procedure* section, that portion of the process plan which describes the manufacturing processes to be carried out. The text of *procedure* sections is divided into lines, or *steps* according to the following alignment rules:

1. the first token is *step number*, which is an integer enclosed with doubly-paired corner brackets;

2. the second token is a system-dependent *work element name*, which designates a process which the workstation can be commanded to carry out, and which is associated with an attribute list;

3. if the attribute list associated with the second token is not empty, then the next set of tokens consists of a left parenthesis, a list of attribute-value pairs of the form:

   attribute-name → *attribute-value*

   and then a right parenthesis;

4. the semicolon line terminator.

The variety of work element keywords and their attribute keywords is system-dependent. However, one particular attribute is required. It is spelled PREC-STEPS, and is that attribute whose value is the list of all of the steps in the *procedure* section which must be successfully completed before the step in question can be started. This requirement enforces an ordering on steps which can be modeled as a directed graph. Although this ordering of steps does not guarantee the intended effects, it does permit certain crude error checks, such as to make sure that the step number is greater than the largest step number in PREC-STEPS.

The purpose of APPI is to determine whether process plans are correct and complete. A process plan is correct if it conforms to the syntax of the AMRF flat file format, and if all of the work element names used in the *procedure* section represent work that can be performed by the targeted workstation. A process plan is complete if all work elements needed to manufacture the part have been provided in the proper sequence and if appropriate values have been assigned to all attributes of the work elements. If the plan is incomplete, the verifier will identify the missing element by issuing either an error message or a warning.

During the March 1987 AMRF benchmark test, the HWS version of APPI was designed to work in the following manner. Flat file format process plans were transmitted to *AMPLE* from HWS using the **Acomm** ACCEPT-FILE keyword (Appendix A). The current file would then be stored locally as an ASCII file called *partname*.in. In response to **Acomm** keyword PROCESS *partname*, APPI would then get the named process plan and build an internal representation of it in *AMPLE/core*. In order to build such an internal representation, a first pass shallow analysis of the plan has to be done. If the first pass is successful, then a more thorough syntactic analysis is performed on the internal representation resulting from the first pass, making use of local archival data which enumerate all legal work elements for HWS and also all legal attributes of each work element.

Some attributes are optional and others may have default values, so APPI does not assume that the attributes of a work element are listed in any particular manner. Thus, the argument list in every step of a process plan must contain both the attribute name and its associated value. During the March 1987 benchmark test, the absence of an attribute merely caused a warning, which implied that all attributes were regarded as optional. This policy has since been changed to allow for the specification of some attributes as mandatory. If the second pass terminates without fatal errors, then APPI returns a validated internal representation of the process plan to *AMPLE/core*, and an ASCII file *partname*.out to ARTCI for further processing. The ARTCI process, which does not require the intervention of any external operator or programmer, returns an ASCII file *partname*.send, as described above. This file will be transmitted to HWS in response to the SEND-FILE **Acomm** keyword. If the second pass does not terminate successfully, then APPI returns the value **nil** to *AMPLE/core* and also returns a file *partname*.err for transmission to HWS.

1. FIRST-PASS ANALYSIS

The following pseudocode, in conjunction with the **Acomm** pseudocode presented in Appendix A, describes the first pass of the syntactic analyzer, called **build-pplan-rep**, which accepts an ASCII file containing a process plan in the flat file format.

```
procedure build-pplan-rep (filename)

begin
  initialize readtable;
  plan-open filename;
  get token from filename;

  if token = --PROCESS_PLAN-- then

    get token from filename;
    if token = --HEADER_SECTION-- then
      build header-rep;
    else
      put "no header section" on error-report;
      plan-close;
      return nil;
    end if;

    get token from filename
    if token = --PARAMETERS_SECTION-- then
      build parameter-rep;
    else
      put "no parameter section" on error-report;
      plan-close;
      return nil;
    end if;
```

```
    get token from filename
    if token = --REQUIREMENTS_SECTION-- then
      build requirement-rep;
    else
      put "no requirement section" on error-report;
      plan-close;
      return nil;
    end if;

    get token from filename;
    if token = --PROCEDURE_SECTION-- then
      build procedure-rep;
    else
      put "no procedure section" on error-report;
      plan-close;
      return nil;
    end if;

    get token from filename;
    if token = --END_PROCESS_PLAN--
        and all sections built then
      make pplan-rep;
      plan-close;
      return pplan-rep;
    else
      put "no end process plan" on error-report;
      close plan;
      return nil;
    end if;

  else
    put "no begin process plan" on error-report;
    close plan;
    return nil;
  end if;

end build-pplan-rep;
```

Comments. The preliminary discussion of the operations to be performed during the first pass should be sufficient to explain most of the operations mentioned above. However, the following points provide additional commentary.

initialize *readtable;*

This operation changes the status of certain tokens in flat file format process plans, and has relevance to the interpretation of the underlying LISP code. (See Appendix B.)

build header-rep;

The **build** operation processes the section one token at a time, continually monitoring its own place within the section and the grammatical structure associated with that place. This process will continue until either an error is detected, in which case **build** returns **nil** to the calling environment; or the appropriate END token is reached, in which case **build** returns the internal representation of the section being processed.

make pplan-rep;
return pplan-rep;

The **make** operation will be executed only if every section of the process plan has been successfully analyzed during the first pass and has been assigned an internal representation by the **build** operation. At this point, the final representation, **plan-rep**, is constructed from the internal representations of all of the sections. The **plan-rep** is then returned to the calling environment. The effect of error detection at this top level is to stop processing altogether and to return **nil**.

2. SECOND-PASS ANALYSIS

If the first pass has successfully terminated, the function **analyze-hws-plan** does the additional job of analyzing the internal representation in greater detail.

```
function analyze-hws-plan (pplan-rep)

  get procedure-section from pplan-rep;
  initialize artci-report;
  initialize error-report;

  loop
    if empty procedure-section then
      close artci-report;
      close error-report;
      return t;
    end if;

    get next step from procedure-section;

    get step-number, prec-steps from step;
    if step-number ≤ maximum of prec-steps then
      put "step sequence error" on error-report;
      return nil;
    end if;
```

```
        get workelement, arguments from step;
        if workelement is not in HWS-symboltable then
          put "unrecognized workelement" on error-report;
          return nil;
        end if;
        put workelement, step-number on artci-report;

        get attribute-list of workelement from HWS-symboltable;

        until empty attribute-list loop
          get next attribute from attribute-list;
          lookup attribute in arguments;

          if attribute is found then
            get attribute, value from arguments;
            type-check value;
            remove attribute, value from arguments;
          else
            put "missing arguments" on error-report;
            return nil;
          end if;
          put attribute, value on artci-report;

        end loop;

        if not empty arguments then
          put "unrecognized attributes" on error-report;
        end if;

      end loop;

    end analyze-hws-plan;
```

Comments. In light of the explanations already given, this pseudocode needs only a few additional explanations.

```
return t;
return nil;
```

The main purpose of **analyze-hws-plan** is to examine the process plan's internal representation which was built in *AMPLE/core* by the first pass, to determine whether the process plan satisfies the conditions on validity imposed by the second pass. If these conditions are satisfied, the Boolean value **t** is returned to the calling environment, indicating that the process plan is correct and complete. If these conditions are not satisfied, then the Boolean value **nil** is returned. The analysis described in this function has no side effects on the representation of the process plan. Aside from the value returned, the only externally visible effects of this function are the files *artci-report* and *error-report*.

workelement **is not in** *HWS-symboltable*

A very important function of *AMPLE/core* is to keep a detailed map of the name space, which is a net of externally visible names and their semantic relations with one another. Nets of this kind may be called *symboltables*. (See Appendix B.) If a work element specification is not found in the *HWS-symboltable*, it is not a legal HWS work element.

type-check ***value;***

In the *AMPLE* prototype of March 1987 the type-checking procedures were preliminary. Since that time a much more extensive type-checking mechanism has been constructed. This mechanism, sketched in [5], will be fully defined in the forthcoming *AMPLE Reference Manual.*

IV. AMPLE REAL-TIME CONTROL INTERFACE (ARTCI)

The *AMPLE* Real-Time Control Interface (ARTCI) is a data preparation system for real-time control systems in automated factories. ARTCI is designed to be a general purpose tool for preparing and verifying control data for real-time control systems of automated manufacturing workstations. This section discusses the implementation of ARTCI for HWS. The ARTCI system provides a user interface for operating its data preparation system, which generates the control data sets for the controllers in the workstation; and for operating its verification system, which may be used for validating and inspecting the data sets.

The verification system is used to test and verify the data generated by ARTCI. The system reads the data files for a particular operation, checks the files for syntax and completeness, and then attempts to execute the files on an internal emulation system. The emulation system is built to match the controllers for which the data are intended, but the outputs of this emulation system are an error and warning report and data for AWAP. AWAP provides the workstation programmer with a view of how the data generated by ARTCI would affect the hardware being programmed.

1. THE DATA PREPARATION SYSTEM

The programming system is the user's window into the ARTCI environment. It provides the programmer with the necessary tools for building control data for a specific part. In this section, a menu system is used to build a general outline of the operations to be performed by the automated factory components. The menus provide general descriptions of sets of operations that are commonly performed with the hardware being programmed. After the outline is built from these data sets, ARTCI asks specific questions about the outline and its effect on the automated factory components in the system. The user's responses to these questions assign values to parameters such as the force required for a particular operation, or the distance which an actuator is required to move. When ARTCI has collected sufficient information to build control data for the specified operations, it proceeds to a second process, data preparation.

The menu system gathers the basic information necessary to build a program for the workstation. It first provides the user with a list of the high-level operations that the workstation is able to perform. These basic operations are selected by the user in the order he wishes them to occur. The menu system then requests information from the user about which MacRow he will use to perform each basic operation. In the case of an automated milling workstation, these basic operations include loading a part, machining a part, refixturing a part, unloading a part to the material handling system, exchanging machine cutting tools, and exchanging fixtures on the machine. There are help windows provided to describe each of the choices to the user and to tell the user what input is expected.

The control data generation process is template-based and generates control data for a specific real-time controller or set of real-time controllers. It also has the ability to generate process plans in AMRF flat file format as described in Section III. The control data generation system has in its data base information about formats for control data, controller functions, and controller capabilities. Based on information from the programming system, the control data generation section of ARTCI builds control data by filling in parameters in

templates. A library of these templates is stored in the local data base. The basic modules of the data generation system are a local data base, the template system, and the control-data-file and process-plan writer.

The local data base is a collection of files containing information about how the workstation operates. This information includes primitive operations for each system component, MacRow definitions based on primitive operations, parameter templates, and templates for writing control data and process plans. In this context, a parameter template is an ASCII string with sections missing. These missing sections are filled in with parameter values from the system and concatenated with the rest of the template to produce a properly formed command string. The menu system uses the MacRow definition files to display MacRow lists and help files. Questions to the user about parameter values are built from parameter templates by the parameter system. Finally, control data and process plan files are built by the control data file and process plan writer using templates from the local data base.

The control-data-file and process-plan writer assembles all of the templates generated by the template system and writes them in the proper form for the various real-time controllers. Process plans are generated by filling in parameters in templates of basic process plan elements. ARTCI currently supports process plans in the AMRF process plan format. The final steps in control data preparation include concatenating multiple files into a single file for the workstation and putting the proper communications headers on all of the files.

```
type FILE-HEADER is record
  total-file-length : 10 bytes;
  status-report : 10 bytes;
  LF-delimiter : 1 byte;
  location-level-0-data : 10 bytes;
  number-level-0-records : 10 bytes;
  location-level-1-data : 10 bytes;
  number-level-1-records : 10 bytes;
  location-robot-data : 10 bytes;
  number-robot-records : 10 bytes;
  location-nc-data : 10 bytes;
  number-nc-records : 10 bytes;
  location-fixture-data : 10 bytes;
  number-fixture-records : 10 bytes;
  location-mbc-data : 10 bytes;
  number-mbc-records : 10 bytes;
  LF-delimiter : 1 byte;
end record;
```

These files are then stored in a local data base under their program name. The following pseudocode describes how the ARTCI system functions:

```
procedure artci (program-name)

begin
  read MacRow-files;
```

```
    if mode is interactive then
      open menu-system;
      while parameter-list not full loop
        query user;
        get parameter;
      end loop;
      close menu-system;
    else if mode is automatic
      open file partname.out;
      read parameter-list;
    else
      error;
    end if;

    build workstation-level-0-command-strings;
    build workstation-level-1-command-strings;
    build workstation-level-2-command-strings;
    build robot-command-strings;
    build fixture-command-strings;
    build nc-command-strings;
    build mbc-command-strings;
    build process-plan;

    if error then
      print error-report;
    else
      build transfer-file;
      build file-header;
      print transfer-file;
    end if;
  end artci;
```

Comments. The following commentary should serve to further explain the software modules of ARTCI.

read MacRow-files;

This function reads initialization files telling the ARTCI system about the characteristics of the workstation for which the data are being prepared. The files contain information about which operations are available at each level of control, and what the structure of the operations is.

build workstation-level-0-command-strings;
build workstation-level-1-command-strings;
build workstation-level-2-command-strings;

All three levels of workstation data are generated by these functions. The operation types provided during symbol table initialization are converted to the appropriate formats by filling in templates and matching sequences of events against pre-defined tables. These functions print a separate file for each set of data generated in the ARTCI program. Though the workstation will ultimately require the data to be in one file in a slightly different format, as explained below, the **build** operation writes multiple files so that the system is easier to understand and debug. If an error occurs, the programmer may view each data set independently and check for errors in content and format.

build robot-command-strings;

This function creates the robot strings that will be sent by the workstation to RCS. These strings are generated by selecting an appropriate template for the operation to be performed, and filling the template with parameters from the local data base. Currently, grip poses and numbers must be checked against those selected by the robot programmer.

build fixture-command-strings;

This function generates two types of fixture data. The first type is the command strings that the workstation will send to the fixturing controller, as defined above. These command strings are generated by selecting and filling in templates. The second type of data generated is the data that the fixturing controller uses to execute the commands. These data are loaded into the fixturing controller prior to program execution and are executed by matching the incoming command to a command list with associated actions. Data for the fixturing controller are generated by selecting commands from an available command list and by filling in templates. The file generated for the fixturing controller is called planname.fixt.

build nc-command-strings;

ARTCI provides only commands that HWSC will issue to the High Level Machine tool Controller. The commands are generated by ARTCI from a list of available commands. ARTCI does not generate NC control code.

build mbc-command-strings;

The Materials Buffering Controller commands are generated by selecting from a list of available commands, and filling in parameters.

build process-plan;

The process plans generated by ARTCI are in the AMRF flat file format. The plans are generated by selecting templates designed for each step of part manufacture, and filling in parameters in the templates with information from the local data base.

print error-report;

This routine is invoked if any errors are detected during the course of generating data, and will generate a file of errors encountered during the data generation procedure.

build transfer-file;

This function reads all of the data files generated by ARTCI, compresses them into one large file, and adds a header record into a file called planname.send for communication to the Horizontal Workstation Controller via the **Acomm** system. The file to be sent to HWSC is constructed in the following manner: the entire file will be ASCII, with line feeds (ASCII 10) used for record delimiters. The data records may be any length, but must be terminated by a line feed.

2. THE VERIFICATION SYSTEM

The major sections of the verification system are the file checking system, the emulation system, and the AWAP interface system.

The file checking system checks the data base to make sure that all of the necessary files for the current program name are present, reads the files into local memory, and checks their syntax and completeness. If any files are missing or incorrect, an error file is created and the verification system terminates. If all the files are of the proper form, the file checking system calls the emulation system.

The emulation system uses the information loaded into memory by the file checking system coupled with information from the local data base to perform an emulation of the control data. The emulator tests to make sure that the files can be executed, and it also creates data files that describe an event timeline for all of the hardware in the workstation. This timeline is based on standard times for operations, and describes what equipment is active at every clock tick.

The AWAP interface is comprised of a module for addressing the motion capability of the pertinent workstation (HWS, in this case), plus a general module for data formatting. The workstation-specific module converts task data from the emulation routine into animation data regarding the duration of each motion within the workstation, the sequence of motions, and pauses. Concurrence of motions is also specified, but otherwise relative timing is not included. The animation data must be derived in this form before conversion to the format accepted by AWAP. The formatting module then organizes the animation data into the group

of *set* files required to drive the AWAP simulation of the workstation task. For a description of set files, refer to Section V of this document.

A central element in the operation of HWS is the state table [1]. Before describing the operation of the verification system, the use of state tables in HWS will first be introduced. A state table is a method for representing control flow. In the case of the HWSC, the table consists of a number of paired rows of data. The first row in a data pair is a row of input conditions. If the current state of the workstation matches the conditions on a row of the state table, then the second row of the pair is executed. The pseudocode declarations below describe the structure of an HWS state table.

```
type LEVEL-2-STATE-TABLE-ROW is record
  level-2-command : 10 byte;
  new-command-status : 10 byte;
  active-status : 10 byte;
  robot-status : 10 byte;
  nc-status : 10 byte;
  fixture-status : 10 byte;
  mbc-status : 10 byte;
  current-state : 10 byte;
  robot-command : 10 byte;
  robot-database-pointer : 10 byte;
  nc-command : 10 byte;
  nc-database-pointer : 10 byte;
  fixture-command : 10 byte;
  fixture-database-pointer : 10 byte;
  mbc-command : 10 byte;
  mbc-database-pointer : 10 byte;
  status-to-level-1 : 10 byte;
  new-state : 10 byte;
end record;

type LEVEL-2-STATE-TABLE is file of LEVEL-2-STATE-TABLE-ROW;
```

The following pseudocode describes how the verification system operates:

```
procedure verify (partname)

begin
  read transfer-file;
  check control-data;
  read level-2-state-table;
```

```
    until new-state = DONE loop
      get level-2-state-table-row;
      get any command from level-2-state-table-row;
      if new command then
        process all commands on level-2-state-table-row;
        build animation-records;
      end if;
    end loop;
    if error then
      print error-report;
    else
      print animation-files;
    end if;
  end verify;
```

Comments. The following comments should clarify how the verification system operates.

read transfer-file;

This function reads the file that is to be sent to the workstation controller. It will return an error if one or more of the data files is missing, or if there is an invalid record in any file.

check control-data;

This function checks the control data for syntax, valid commands, and legal parameter values. If any incorrect syntax is discovered, or if an unrecognized command or out-of-bounds parameter is found, an error is returned.

read level-2-state-table;

This function examines the control data and checks to see which MacRows or state tables will be required to manufacture this part. It then loads the state tables into the controller emulator. This function will return an error if the control data specify a state table that is unknown to the system.

get any command from level-2-state-table-row;

This is the main computational routine of the verification system. It keeps track of the simulation's internal states, and checks the state tables for matching rows. If a match is found, internal states are updated and all commands on the state table row are processed.

process all commands on level-2-state-table-row;

This operation generates data tables that will be used to drive the AWAP animation system. This operation has the ability to simulate each controller in HWS, and will simulate the execution of each command the controller receives. **Process** checks to see that the specified controller can execute the given commands in its current state. If the controller can execute the command, animation data are generated to represent the motion the controlled device would exhibit if it were to receive such a command.

build animation-records;

This operation constructs animation records from templates that simulate the commanded actions discovered by process commands.

print error-report;

This function generates an error report file to tell what errors, if any, were found by the verification system.

print animation-files;

This **print** operation prints a set of files sufficient to drive the AWAP animation system. The form of these files is documented in Section V.

V. AMPLE WORKSTATION ANIMATION PACKAGE (AWAP)

AWAP, the *AMPLE* Workstation Animation Package, is a software system for computer graphics simulation of three-dimensional objects in motion. The AWAP software design is not intended for creating the most visually impressive displays, but for creating a visual representation with such accurate movement capability that a user of AWAP can easily predict and study the motion of a physical system without needing to experiment with the actual system. The design of AWAP is sufficiently general to accommodate a wide variety of applications [3]. In the prototype application discussed in this report, AWAP is featured as an *AMPLE* module.

In the course of developing a graphics simulation package, one of the first important objectives was to adopt fairly simple technique for characterizing and recording geometric representations of three dimensional shapes. Since the generation and verification of NC part programs is outside the scope of the *AMPLE* implementation described here, there is no concern for part attributes, and the depiction of parts (workpieces) may be especially plain. A polygonal approximation of the shapes of objects was judged to be sufficient for the immediate needs of the system, and could be adaptable to future needs. Even though a solid representation would be better suited for such capability as interference checking,[3] the mathematical complexity of a solid representation would prohibit animation at an acceptable speed. The polygonal format chosen works well for visually checking the accuracy of motion, and can be converted to a boundary representation of a solid model if the system is to be used with a solid modeling package which accepts a boundary file. The following pseudocode shows how a hierarchical structure of entities makes up a graphics *object*.

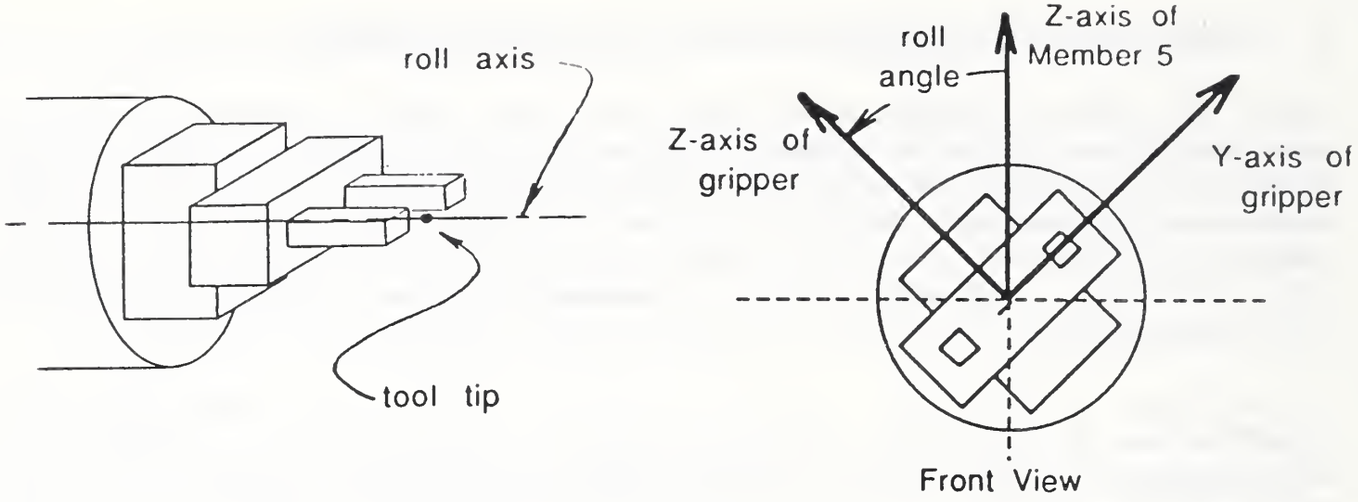
```
type POINT is array(3) of float;
type POLYGON is set of point;
type MEMBER is set of polygon;
type COMPONENT is set of member;
type OBJECT is set of member;
```

Objects are modeled as collections of polygons which are defined as ordered sets of vertices. A crude representation of an object may be made up of very few polygons. For a more accurate model an object may need to be subdivided into more polygons. Depending on the graphics system used for implementation, the use of a very large number of polygons can restrict the speed of animation. In the case of the HWS prototype, as well as that for the AMRF Turning Workstation, precise modeling of most components has not been necessary, and the number of polygons has not been a problem.

Each major component (such as a robot or machine tool) of a modeled workstation requires its own file of geometric data. The data for all the articles to be handled by the robot may be

---

[3] During animation, the polygonal model which is used does not check for interferences. However, AWAP does have the capability of creating a static solid model which does allow interference checking. The section on Viewing Routines explains that animation can be suspended at any moment, and that the GMS function may be used to create a solid model of the workstation in the suspended state.



Robot Gripper

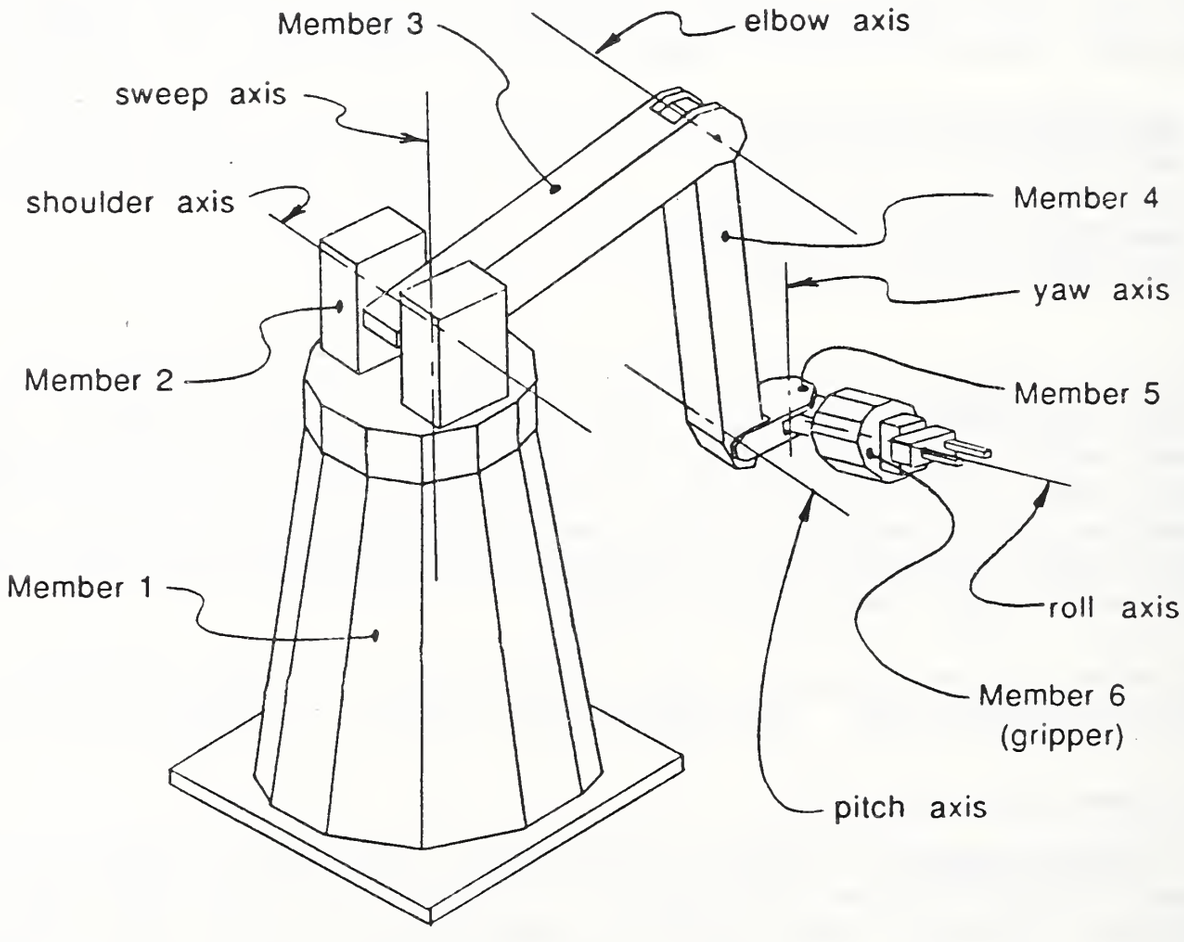


Figure 2. HWS Robot (Component 2)

in a single file. Detailed explanations of how objects are modeled and the format of the geometry files are explained in Bandy [4].

In addition to the data for describing the geometry of the objects, there must also be data for describing the positioning of the objects. For convenience, certain terminology referring to groupings of geometry has been adopted. The term *component* refers to any major component of the workstation such as a robot or a machine tool. The term *member* refers to a rigid body -- a collection of geometric entities with fixed spatial relationships with one another. Components are usually made up of several members (Figure 2). Each component and member has its own assigned Cartesian coordinate system. Graphics objects are defined for convenience. While a member or collection of members may constitute an object, the component of which those members are a part may also be an object. Each object also has an assigned coordinate system.

AWAP also features a system of animating the acquirement and placement of articles by a robot. The task of specifying the range of possibilities for handling articles is simplified with the use of coordinated sets of data structures. The grip points and the desired orientation of an article are known in terms of the coordinate system of the article. Each location in the workstation in which an article may be placed is characterized by the coordinate system of that *Article Location* and by identification of the surfaces against which an article may be seated. At all times, data management routines keep track of the locations and orientations of each Article Location and of each article within an Article Location, with respect to all pertinent coordinate systems. This design allows the pick-and-place commands to be very simple. Instead of having to keep track of coordinates, the user uses commands involving only such specifications as approach vectors, target article or target location. Adjustment of robot gripper width to suit the article to be grasped is automatic.

1. THE HWS PROTOTYPE OF AWAP

One of the prototype applications of AWAP has been for a visual verification of control data for HWS. In this application, data for operating the equipment in the workstation are first used to drive an AWAP animation of the workstation in operation. This allows a user to observe in advance the gross motions of the workstation components in response to the control data.[4] If the animation of the workstation reveals actions which are not desired, it is known that the control data contain mistakes which can then be identified and corrected. The AWAP simulation can then be run again to validate the modified data.

1.1. A Segmented Perspective of HWS

In the modeling structure used here, each major device in the workstation, as well as each *article*, is classified as a *component*. A complete list of every HWS *workstation-component* follows.

---

[4] Note that the response of the workstation to conditions which cannot be predicted on the basis of the control data will not be simulated.

```
type WORKSTATION-COMPONENT is
 {robot,
  machining center,
  vise-fixture,
  v-block-fixture,
  active-pedestal,
  stationary environment,
  tool,
  part}

type ARTICLE is
 {tool,
  part}
```

This section of the report regards HWS in a slightly different context than Section 2. For the purpose of developing kinematic models, the workstation must be considered in terms of its components or subcomponents which move with respect to others. The workstation consists of a six-axis industrial robot, a numerically controlled (NC) horizontal machining center, an automated fixturing system, two robot re-grip stations, and a tray buffering device for storing, loading and unloading trays of parts, blanks, and tools from a robot cart delivery system. The robot has the ability to change end effectors (grippers) as needed, selecting from grippers mounted on the robot base. The machining center has two pallets on which fixtures to hold workpieces are mounted. Each pallet may rotate about a vertical axis, and may shuttle between its loading position (either at an extreme left or right position as one faces the machining center) and the machining position. A tool drum which may hold up to thirty tools operates such that the robot may load or unload tools, and such that the changing of tools for machining may be done automatically by the machining center. The stationary re-grip station is a table on which the robot may place an article and then grip the article in another manner. Another re-gripping station is the active pedestal, which rotates the article before it is again grasped by the robot. The tray buffering device presents trays to two locations in the robot work space.

1.2. Implementation Environment

The HWS edition of AWAP, written in FORTRAN, is designed to be ported to many different computers or types of graphics systems. Although development of the HWS prototype started with a Ramtek 9400 graphics system hosted by a VAX 11/780 computer, the current implementation uses the Silicon Graphics IRIS computer and graphics system described in Section 1. The IRIS provides sufficient speed for the real-time animation desired for the AWAP system. A convenient feature of the IRIS system is its manner of defining a graphics *object* as a group of geometric entities which are all subject to the same transformations. Thus defined and treated, these objects are ideal for partitioning a mechanism for kinematic modeling [12]. The AWAP prototype makes use of the easy drawing and graphics manipulation commands available through the IRIS Graphics Library's FORTRAN callable subroutines. The prototype of AWAP also takes advantage of the Z-buffering hardware feature to create static images with hidden surface removal.

Another tool which facilitated the development of the prototype is the *AMPLE* Coordinate Editor [4]. The Coordinate Editor is a software package which may be used to create geometry files for workstation components to be simulated. According to the known dimensions of a component, a user must visualize the component as consisting of three-dimensional primitives [10] such as boxes, cylinders, cones, etc. The Coordinate Editor creates these primitives and automatically writes the corresponding polygon vertex coordinates into a file in the required format. The resulting files of geometric data are read and interpreted by AWAP to create the graphics representations of the workstation.

2. THE HWS EDITION SOFTWARE

Before going into detail concerning the HWS implementation of AWAP, an overview will first be presented in pseudocode form. Then the balance of the report will elaborate on the design and operation of the package.

2.1. Summary of the HWS AWAP Software

Following is an overview of the HWS Prototype of AWAP.

```
procedure animation-sequence (set files)

begin
  initialize graphics system;
  load color map;
  get coordinate-data;
  get initial-positions;
  create objects;
  specify default view;
  create screen layout and fixed images;

  until keyboard = 5 loop
    if keyboard = 4 then
      modify set-file with set file editor;
      reset current set file pointer;
    end if;

    until final set-file loop
      get set-file;
      get motion-specs from set-file;
      convert motion-specs to positioning-specs;
      calculate modeling-matrix of each member for each animation frame;
```

```
            until final frame count loop
              until final member loop
                get modeling-matrix;
                display-matrix ← inner product of modeling-matrix and view-matrix;
                calculate arguments for graphics-calls;
                execute graphics-calls;
              end loop;

              if keyboard = s then
                until keyboard = f loop
                  get view-options;
                  modify view;
                end loop;
              end if;

            end loop;

          end loop;
        end loop;
      end animation-sequence;
```

2.2. HWS Kinematic Models

The positioning of a workstation component is implied by a parameter list used to characterize the positioning of the *members* (as defined above) of that component. The constituent polygons of a member are defined in terms of that member's own coordinate system. The positioning of a member is determined by the translation and rotation of its coordinate system with respect to some other specified coordinate system. Each member has certain positioning constraints. Some constraints locate the coordinate system of the member with respect to another coordinate system, while other constraints impose orientation limits. The number of parameters necessary to uniquely define the positioning of a group of members is equal to the degrees of freedom of the group, minus the number of applicable constraints. Although the number of parameters used to define the positioning of a group of members is consequently fixed, the actual parameters may be chosen for convenience. Such a list of parameters satisfies the requirement for positioning data. Kinematic equations which consider the parameters and constraints for each group of members determine the unique positioning of each member.

The kinematic model of a component is the set of equations for determining all the transformation parameters (such as angles of rotation or distances of translation) for each moving member of that component, as a function of less explicit positioning specifications. If the motions of the members are independent of one another, as was the case for most of the components in HWS, then the kinematic equations are simple. On the other hand, the robot required a reasonably complex kinematic model.

The six degrees of freedom for the HWS robot are controlled by the six revolute joints. Actuators at these joints are the means for the actual robot to attain any position. It is also in terms of these joints that the positioning of the graphics representation of the robot is

defined. However, in actual use, the joint angles would seldom be known. The practical specification for robot position is the location and orientation of the gripper. Such a specification of gripper positioning, along with the known constraints of the robot design, can be used to uniquely determine the six joint angles in practically any case [15]. (There are a few unlikely cases in which an infinite set of joint angles would satisfy the type of conditions described here.) The kinematic model was designed to calculate the six joint angles as a function of several types of positioning specifications.

2.3. Robot Positioning Specifications

The method for controlling the simulation of the robot is more complex than that for any other component of the workstation. For this reason, the treatment of the robot will be described in detail here, with the expectation that the method for controlling any other component will become obvious.

The positioning specifications used by AWAP were selected conveniently and uniquely define any positioning which may actually occur in the workstation. It is important to note that the positioning specifications used by the version of AWAP described in this report are not meant to be the exact specifications used with the actual workstation equipment. This AWAP prototype is only meant to demonstrate possibilities. With this understanding, the following approach was taken for the robot.

An attempt was made to simplify the task of specifying desired positions of the robot. Three methods of doing so are described in this section. (The actual robot position parameters required by the graphic model are explained in the Principles of Animation Control section of this report.)

Methods of specifying robot position:

1. Any position of the robot is uniquely defined by the six joint angles. Although AWAP accepts such a specification, these angles are not usually known by the user.

2. When the transformation matrix of the article to be gripped is known, the desired matrix for the gripper in the corresponding position may be easily derived. If the position is physically attainable, the parameters to define the positioning of the rest of the robot may then be calculated.

3. If the location and orientation of the article to be gripped are known in terms other than a transformation matrix, it may be most convenient to first specify the desired location of the gripper tool tip, then specify the desired orientation of the roll axis (Figure 2), and then specify the requirements for the roll angle. The position of the gripper is thereby defined, and if the position is attainable, the parameters to define the positioning of the rest of the robot may then be calculated.

Any of the above three methods may be used to specify the position of the robot. As explained in the Principles of Animation Control section of this report, positioning requirements related to the handling of articles are specified in terms of the articles, so the

above considerations are taken into account automatically by AWAP software. For robot positions to be specified without regard to any article to be handled, the third method of specification has been found to be most useful. Therefore roll axis orientation becomes an important concept.

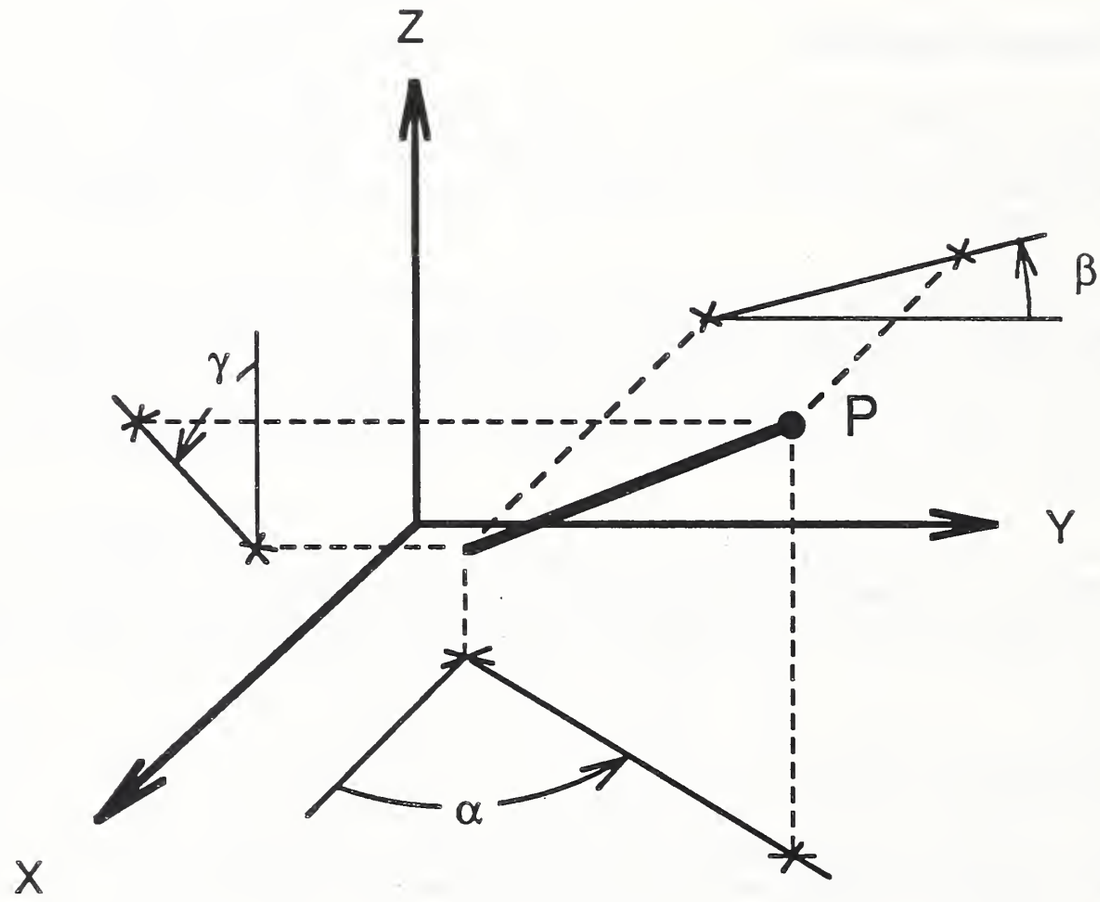


Figure 3. Gripper Orientation

The bar in this figure represents the roll axis of the HWS robot. Point P represents the tool tip. Therefore the gripper orientation (without regard to the roll angle) is the same as that of the bar. Angles α (alpha), β (beta) and γ (gamma) are the angles between the coordinate axes and the projections of the bar onto the coordinate planes, as shown. The orientation of the bar is uniquely defined by any two of the three angles. Note, however, that one of the three angles is undefined if the bar is parallel to one of the coordinate axes.

While there are many possible ways of specifying roll axis orientation and roll angle, the following methods are judged to be convenient for our purposes. The orientation of the roll axis may be designated in either of the following two ways:

- Give the components of a unit vector pointing in the positive X-direction of the local coordinate system of the gripper (i.e., pointing through the arm along the roll axis towards the tool tip. See Figure 2).

- For each of two coordinate planes, give the angle between the projection of the roll axis onto the plane and the lower coordinate axis. (See Figure 3.) The lower coordinate axis of a coordinate plane is the first of the two axes identifying a coordinate plane according to right-hand rotational order (X, Y, Z, X, Y, etc.). For example, the lower axis of the Y-Z plane is Y; the lower axis of the X-Z plane is Z.

The roll angle may be given explicitly if known. Otherwise, requirements may be designated in either of the following two forms:

- Give the components of a unit vector pointing in the positive Z-direction of the local coordinate system of the gripper. (See Figure 2.)

- Indicate whether the fingers of the gripper are to be in a vertical plane, or whether the fingertips are to be in a horizontal plane. For uniqueness, also indicate whether the smallest roll rotation to achieve this position would be clockwise or counterclockwise. Note that this type of designation is only possible in special cases.

The requirements for the roll angle, together with the orientations of the roll axis and the local Z-axis of Member 5 (Figure 2), comprise enough information to calculate the roll angle.

The above method of having the roll axis orientation as an independent specification is favored over other possibilities because of two major advantages. First, the method does not require the user of AWAP to derive a single specification that implies the orientation of the gripper, a three dimensional object. The second advantage is that the sixth degree of freedom of the robot -- the roll angle -- becomes independent of the others, simplifying the kinematics problem. Instead of six simultaneous nonlinear equations, there are five. The solution was appropriately formulated and incorporated into the AWAP software. Details on the development of HWS kinematic models are explained in Bandy [2].

3. VIEWING ROUTINES

To simulate motion, modeling matrices govern changes in the graphics image of the workstation components. However, for the image to be displayed in the desired manner, the workstation coordinates which would result from the modeling matrices alone must be scaled, rotated, transformed from three to two dimensions, and clipped. These effects are contributed by view matrices. The matrix product of the modeling matrix and the view matrix is the display matrix which is directly used to create the image on the graphics screen. The viewing routines which have been developed to suit the needs of AWAP manipulate the following matrices.

```
type MODELING-MATRIX is array(4,4) of float;
type VIEW-MATRIX is array(4,4) of float;
type DISPLAY-MATRIX is array(4,4) of float;
```

The different viewing functions are controlled by the user's interaction with the IRIS *keyboard* and *mouse*. Characters resulting from keystrokes are stored in queues until read and processed.

```
type KEYBOARD is queue of character;

type MOUSE is queue of character;

type VIEW-OPTIONS is
 {color,
  default,
  gms,
  identify-member,
  near-far-clipping,
  orthographic-views,
  pan,
  rotate,
  zoom}
```

AWAP allows the user to view the display of the workstation in practically any manner possible. The user may suspend animation at any point to exercise the viewing capability. Viewing functions include:

Color Priority

The conversion of the display from a *wireframe* appearance to an image with hidden surface removal. The image seen during animation is wireframe with no hidden surfaces removed, even though in the default view the priority of the colors was chosen to give the appearance that the foremost workstation components are in front of the others. When animation is suspended and the Color Priority function is used, AWAP first counts the number of bit planes that are in the IRIS processor. If there are fewer than thirty-two bit planes, all the polygons which make up the entire workstation are sorted according to how far they are from the user, and are drawn and filled in order, from the furthest first to the nearest last. If the system has thirty-two bit planes, AWAP automatically executes a more accurate method of hidden surface removal: the IRIS Z-buffering function.

Default

A specific view of the workstation, selected such that a single image of the entire workstation fills the viewport and permits the viewer to see the top, front, and a side all at once.

GMS

A function which writes instructions for creating a solid model of the workstation in the same state in which animation was suspended. GMS, a commercial solid modeling package, may subsequently be used to draw the workstation in that state.[5] In addition to other uses of the GMS representation, the workstation may be viewed with color shading (Gouraud).

Identify Member

A feature which allows the user to identify any desired *member* in the workstation by causing the color of that member to flash.

Near-Far Clipping

A function for moving the near and far clipping planes by moving the IRIS mouse and pressing mouse buttons. The default view situates the clipping planes so that the near plane is nearest to the viewer, the far plane is therefore further from the viewer, and the complete image is between the near and far planes. By moving the far plane or the near plane into the image, only the portion of the image remaining between the planes may be seen, and the rest of the image is *clipped* -- not displayed. When controlled, this effect may be used to suppress the drawing of portions of the image which are not of immediate interest, thereby producing a less cluttered display.

Orthographic Views

A function allowing the viewer to see any one of the following views: front, top, or side.

Pan

The function for controlling the movement of the image in any directions parallel to the plane of the display screen. The appearance is given that the viewer is *walking by*. The user may control this function by moving the IRIS mouse.

---

[5] GMS is the Geometric Modelling System, a trademark of Interactive Computer Modelling, Inc.

Rotate

The function which may be used to control the orientation of the entire image on the display screen. The appearance is given that the viewpoint is moving around, over or under the image. By moving the IRIS mouse to the left or right, the user causes a positive or negative rotation, respectively; and the angle of rotation will be the azimuth, incidence or twist, depending on which mouse button is held down.

Zoom

The viewing function allowing the user to control the scale of the image so that it becomes either larger (*zooming in*) or smaller (*zooming out*) at the same time that the workstation coordinates at the center of the display screen remain at the center. Whether the zooming is in or out depends on which IRIS mouse button is held down.

The *AMPLE* Coordinate Editor, the program for describing the geometry of the workstation components, uses the same viewing routines as the AWAP implementation described in this report. A complete explanation of the viewing routines used in the HWS prototype is presented in Bandy [2].

4. PRINCIPLES OF ANIMATION CONTROL

This section shows how data are specified for creating desired motions. The prime intention here is to present examples of data formats for performing specific actions, and thereby illustrate the principles of AWAP animation. The examples will describe the different types of data whose values must be reassigned from one animation sequence to another, and to state the requirements for use of those data. However, the locations of files containing other data will be mentioned first to establish the context of the discussion. It is important to note that the HWS implementation of AWAP is intended only to demonstrate the potential of the package. Neither the geometric representations nor the animated behavior are meant to accurately model HWS.

4.1. Locations of Related Files

In the HWS prototype of AWAP, files were organized into directories which are designated for particular purposes. To organize the discussion of data, files will be referred to as belonging to the same directories as in the prototype. The directory structure is illustrated in Figure 4. The main directory from which the animation package will be controlled will be referred to as *Main*. (For the *AMPLE* system on the IRIS, Main may be equated to /d/ample.) The source files are located in Main/Hws. Although the executable image is actually located in Main/Execut, the program must be executed by running the UNIX shell script Main/Hws/hsnarc. The geometry files are located in Main/Data/Cdata. There is one geometry file for each major component (robot, v-block fixture, etc.). Geometric data for all articles (tools and workpieces) are contained in a single file: harticl.dat. (The geometric data format is explained in Bandy [4].)

In harticl.dat, articles must be numbered in integer sequence, the first being Article 10. It is best for harticl.dat to contain data for no more articles than will be necessary for anticipated animation. Timing of animation is most accurate when the program has to keep track of the fewest articles. But even if 12 (the maximum) articles are used, the worst effect would be a ten percent slower frame rate and a pause of up to five seconds between sets. (i.e., as opposed to about two seconds between sets if no articles had been read. See the Animation Commands section for an explanation of *sets*.)

The data which determine the initial positioning of all the articles in the workstation are contained in Main/Data/Hdata/hartic.dat. The initial positions of everything else in the workstation are defined by the parameters in Main/Data/Hdata/hparam.dat. The files of data which govern the motions in an HWS animation sequence are located in the directory /d/ample/Data/Hdata, and must have the filenames set.1, set.2, ..., set.n. The file containing the captions to be shown with each set file is Main/Data/Hdata/hcode.set.

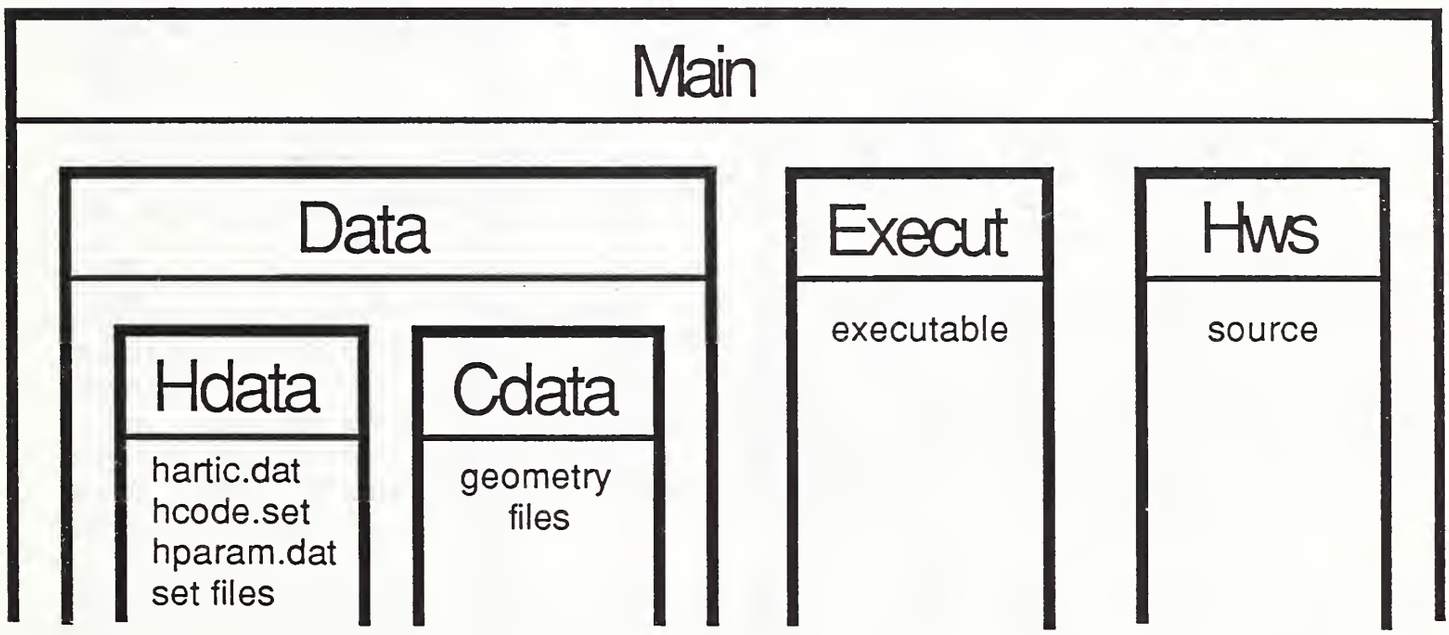


Figure 4. Directory Structure

Only pertinent files and directories are shown. The directories shown actually contain additional files and subdirectories in support of other AWAP editions (TWS, etc.) and the Coordinate Editor.

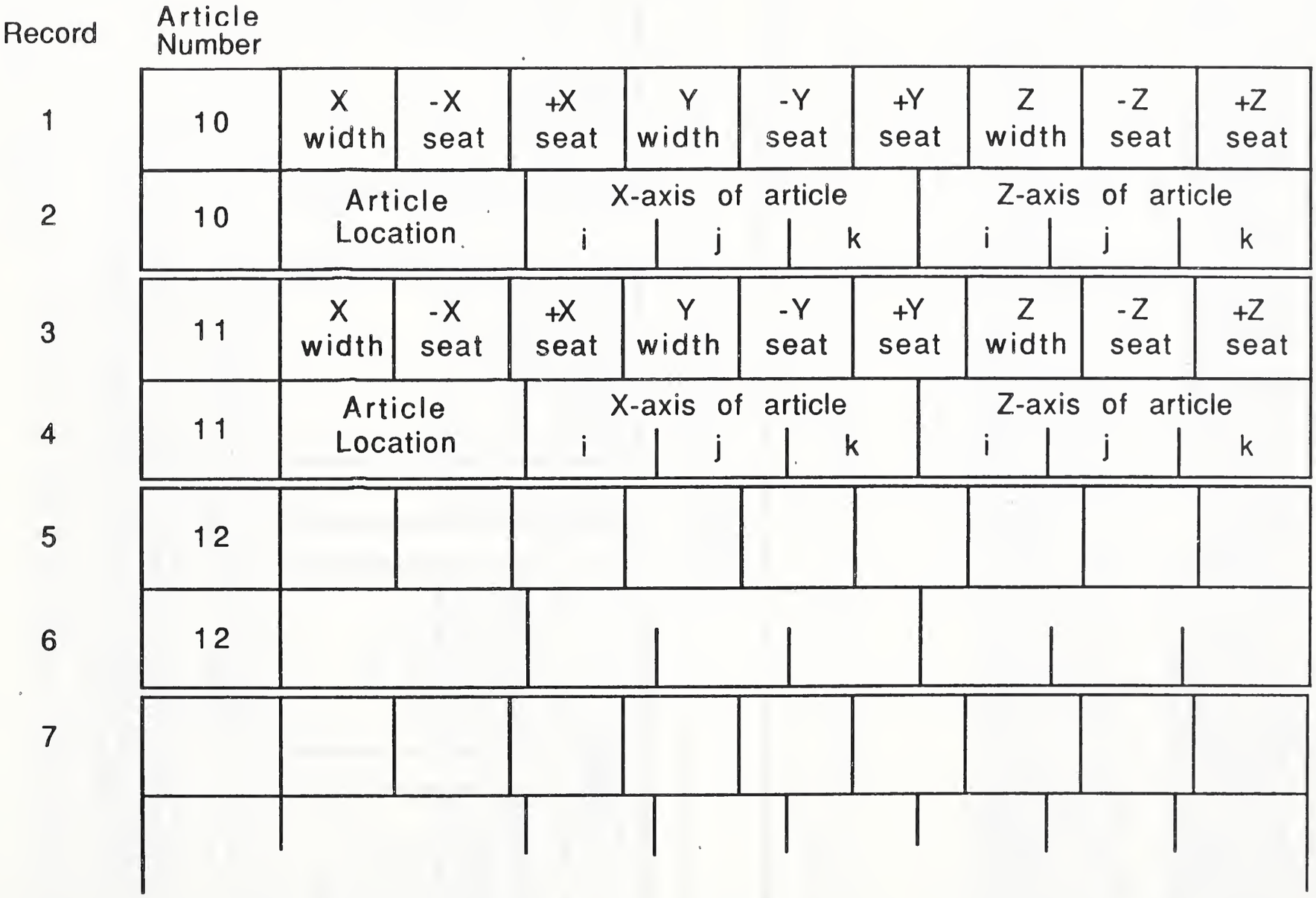


Figure 5. Data Elements in File hartic.dat

4.2. Initial Positions of Articles

The initial positions of articles are specified in File hartic.dat in terms of orientation and Article Location. Applicable Article Locations are:

200. Kardex Tray #1
201. Kardex Tray #2 (nearer to machine tool than Tray #1)
202. Vise Fixture
203. V-Fixture, front load
204. V-Fixture, top load
205. Active Pedestal
206. Regrip Table
207. Tool Drum
208. (not applicable)
209. (not applicable)
210. (not applicable)
211. Robot Gripper
212. (not applicable)
213. (Kardex) Hidden Buffer

The above Article Locations are to be used only in hartic.dat. As explained in the Animation Commands section, different location codes called World Locations are used in set file commands.

The orientation of an article is defined in terms of world space components of unit vectors in the directions of the X and Z axes of the article coordinate system. (In world space, +X is from the robot towards the machine tool, and +Z is up.)

File hartic.dat consists of a pair of consecutive records for each article, as shown in Figure 5. The first record for each article contains permanent data regarding the gripping dimensions of the article. Every second record contains data specifying the initial positioning of the article. The sizes of the data fields do not matter since the file is read by a list-directed mode of access. The data elements in the second record for each article appear in the following order: Article number, Article Location, i, j and k components for the X-axis of the article, and i, j and k components for the Z-axis. The article number is of integer data type; all other data elements are real.

Although the remainder of this section is essentially unrelated to the initial positions of articles, it is necessary to complete the explanation of File hartic.dat. Permanent data for articles -- i.e., data that do not change with the animation sequence -- are stored in the odd-numbered records of hartic.dat. The permanent data are dimensional information for seating articles in Article Locations. A *seating surface* is a surface of the article which may contact some surface of a location in which the article may be placed. For each article, the following data are permanent:

```
10      1.00   -.50    .50     1.00   -.50    .50     5.00  -2.50   2.50
10       200.     .0000     1.0000     .0000     .0000     .0000    1.0000
11      4.00  -2.00   2.00    2.00  -1.00   1.00    3.00  -1.50   1.50
11       201.     .0000     .0000     1.0000     .0000     1.0000     .0000
12      2.00  -1.00   1.00    2.00  -1.00   1.00    5.50  -2.75   2.75
12       213.     1.0000     .0000     .0000     .0000     .0000     1.0000
```

HARTIC.DAT (example)

```
 4      0.      1.      11.0   -1.0000    .0      .0      .0      .0     1.0000
50      0.      1.       .0      .0       .0      .0      .0      .0      .0
 4      2.      5.      11.0     .0       .0      .0      .0      .0      .0
50      2.      5.       .0      .0       .0      .0      .0      .0      .0
 4      6.      7.       .0      .0       .0      .0      .0      .0      .0
```

SET.1 (example)

```
 7      4.      5.       3.0     .0       .0      .0      .0      .0      .0
 4      0.      5.     102.0   -1.0000    .0      .0      .0      .0     1.0000
 4      6.      9.     102.0    11.0      .0      .0      .0      .0      .0
 7     10.     11.       2.0     .0       .0      .0      .0      .0      .0
 3     12.     14.       2.0    11.0      .0      .0      .0      .0      .0
50     12.     14.       .0      .0       .0      .0      .0      .0      .0
```

SET.2 (example)

```
' Carry part to vise fixture'
' Insert part in vise fixture'
```

HCODE.SET (example)

Figure 6. AWAP Data Files

- Grip widths parallel to each axis of the article coordinate system
- Distances from the origin to the seating surface of the article in each of the six axis directions. (The current implementation requires that the distances to the two seating surfaces normal to an axis be equal. Each article is defined with its origin at the midpoint between its grip points.)

Each odd-numbered record in hartic.dat contains the following data elements, respectively. (See Figure 5 and the hartic.dat example in Figure 6.) As with the even-numbered records, the sizes of the data fields do not matter.

- article number (integer)
- grip width (real) parallel to the X-axis of the article
- seating surface coordinate (real) in the -X direction
- seating surface coordinate (real) in the +X direction
- grip width (real) parallel to the Y-axis of the article
- seating surface coordinate (real) in the -Y direction
- seating surface coordinate (real) in the +Y direction
- grip width (real) parallel to the Z-axis of the article
- seating surface coordinate (real) in the -Z direction
- seating surface coordinate (real) in the +Z direction

4.3. Initial Positions of Workstation Components

The initial positions of all workstation components except articles are determined by parameters contained in the file Main/Data/Hdata/hparam.dat. Where lengths are directly specified, the unit of length is the inch. The sizes of the data fields in hparam.dat do not matter; nor do the number of values per record, since the file is read by a list-directed mode of access. The data types of the initial position parameters are indicated in the following pseudocode.

```
type DEGREES: float;

type INITIAL-POSITIONS is record
  gripper-code: integer;
  finger-width: float;
  gripper-orientation: array(3) of float;
```

```
    tool-tip-location: array(3) of float;
    roll-angle: array(3) of float;
    vise-jaw-opening: float;
    left-pallet-y-coordinate: float;
    left-pallet-rotation: degrees;
    right-pallet-y-coordinate: float;
    right-pallet-rotation: degrees;
    quill-x-coordinate: float;
    spindle-z-coordinate: float;
    tool-drum-position: integer;
    v-block-jaw-opening: float;
    active-pedestal-jaw-opening: float;
    active-pedestal-rotation: degrees;
    reserved-parameters: array(7) of integer;
end record;
```

Brief explanations of the initial position parameters follow. Where commands are mentioned, refer to the Animation Commands section.

The *gripper-code* indicates which gripper is mounted on the robot wrist, and is the same code used in the CHANGE GRIPPERS command. The *finger-width* is the distance between the fingers of the gripper mounted on the wrist. The *gripper-orientation* may be specified according to the same requirements as the first three parameters of the GO XYZ (ijk) command, or optionally according to the same requirements as the first three parameters of the GO XYZ (ANG1ANG2) command. The *tool-tip-location* is comprised of the X, Y and Z coordinates of the tool tip of the gripper, in terms of the robot coordinate system -- i.e., the coordinate system parallel to the world system, but having its origin at World [0., 0., 59.42]. The *roll-angle* is specified according to the same requirements as the first three parameters of the ADJUST ROLL ANGLE command.

The *vise-jaw-opening* is the width between the jaws of the vise fixture. The *left-pallet-y-coordinate* may locate the machine tool pallet containing the vise fixture anywhere in the range between the machining position and the loading position. *left-pallet-rotation* is the angle of rotation (about the Z-axis), in degrees, of that same pallet. The *v-block-jaw-opening*, *right-pallet-y-coordinate*, and *right-pallet-rotation* are similar to the data for the vise fixture and left pallet, but apply to the v-block fixture and right pallet.

The *quill-x-coordinate* and *spindle-z-coordinate* locate the in-out and up-down positions, respectively, of the spindle of the machine tool. The *tool-drum-position* is the number indicating which tool is in position to be transferred to the spindle.

The two active pedestal parameters are *active-pedestal-jaw-opening*, indicating the width between the inner faces of the jaws; and *active-pedestal-rotation*, the angle of rotation, in degrees, about the Z-axis. The seven final *initial-positions* elements, the *reserved-parameters*, are not used, but exist to allow the possibility of adding new capability without revising format.

The above parameters are identified by sequence numbers 1 through 29, in the order that they appear in the pseudocode. Also contained in hparam.dat are Parameters 30 through 129, comprising the default first set of animation data, to be used in the event that Main/Data/Hdata/set.1 does not exist. Therefore the data for these parameters must be in accord with the requirements of animation data, as explained in the next section. Only Parameters 30 - 39 are required to be data for a valid *set-file* command (explained below). The remaining values may be zeros. The data types are integer for parameters 30, 40, 50, 60, 70, 80, 90, 100, 110, and 120; and are real for all others.

4.4. Animation Commands

The data which drive the animation are organized as shown in the following pseudocode.

```
type PERIOD is record
  group: integer;
  start-time: float;
  end-time: float;
  position-parameters: array(7) of float;
end record;

type MOTION-SPEC is set of period;
type SET-FILE is sequence of motion-spec;
```

The data files, called *set* files, that govern the motions in an HWS animation sequence must be located in the directory Main/Data/Hdata, and must have the filenames set.1, set.2, ..., set.n. Each record in a set file represents a *period*, a time interval in which action for a specified motion group takes place. A *motion group* (or just *group*) is a collection of geometric entities whose changes in position depend on a common set of equations. A collection of periods constitutes a *motion-spec* -- an animation command which implies motion for the designated motion group. The start and end times of each period of each motion-spec are measured in seconds from the start of the set file. Time intervals for different periods are permitted to overlap if different groups are specified for each. Periods are referred to as the sequence numbers of the records in the set file. *Position parameters* define the end-time positioning of the geometric entities represented by the group number. Figure 6 contains two examples of set files.

The sizes of the data fields do not matter since the file is read by a list-directed mode of access. The following rules apply to set files:

1. The order of occurrence of any one group in a set file must be chronological. Although no two periods for the same group may have overlapping timing, the start time for one occurrence may be the same as the end time for the last. (The exception for *dummy* periods is noted later.)

2. The maximum duration of a set file -- that is, the greatest end time that can be specified -- is sixty seconds.

3. The maximum number of periods in a set file is twenty.

When a set file is executed, the data for each period cause a gradual change in the positioning of the geometric entities of the corresponding group. At the start time, the positioning starts to change from its previous state such that the positioning implied by the parameters is achieved at the end time. Since set files are processed and executed sequentially, animation will pause, perhaps for about two seconds, between set files. To mark the end of an animation sequence, the last set file of the sequence must contain only a dot (ASCII 46).

There are a few commands whose set-file elements deviate from the preceding description. For example, there are certain cases for which the end positioning does not occur at the specified end time. The explanations of set-file requirements for each command, given in the rest of this section, will note such exceptions.

The following few sections contain examples of animation command formats. Certain information required for each command will be identified to illustrate the manner in which the animation is controlled. Since the intention is not to present a complete instruction manual, information considered to be irrelevant to a general understanding will be either omitted or abbreviated.

The position parameters required for the principal period of each command will be indicated. As shown above in the type definition for *period*, there must be seven parameters per period; so if fewer than seven are indicated, then the remaining parameters are constants associated with the command. (The values of such constants are beyond the scope of this report.) All vectors are represented by three floating point values -- the normalized i, j, and k components. For the cases in which the first period for a command must be followed by one or more dummy periods, each dummy period should have the same start and end times as the first period for the command, but all dummy-period parameters should be equal to zero. Dummy periods must appear in the same set file as the initial period for the command with which they are associated.

Note that descriptors -- worded expressions such as *GO TO XYZ (ijk)* -- are used to refer to the animation commands. The descriptors themselves are not used as commands, but are only names identifying commands. As shown below, the actual commands are in the form of sets of numerical values.

4.4.1. Independent Robot Commands

These commands are for the robot to move without regard to any article. The *robot coordinate system* referred to is parallel to the world coordinate system, but has its origin at World [0., 0., 59.42]. When the robot moves its gripper from one location and orientation to another, the tool tip follows a linear trajectory at the same time that the attitude of the gripper also changes in a linear manner. If any joint limit is reached during an attempt to traverse a linear trajectory, the tool tip deviates from the idealized trajectory only to the extent required to prevent violation of the joint limit. If a destination specification is unattainable by far, then the results are unpredictable. (The HWS robot algorithms are explained in Bandy [2].) Figure 2 shows the basic configuration of the HWS robot.

GO TO XYZ (ijk). Three parameters denote the roll axis orientation vector -- i.e., the vector along the roll axis, in the direction of the tool tip -- in terms of the world coordinate system. Three other parameters are the coordinates of the tool tip in terms of the robot coordinate system. The robot moves the gripper to the specified location and orientation. The roll angle will remain as before unless the ADJUST ROLL ANGLE command is simultaneously issued.

GO TO XYZ (ANG1ANG2). This is an alternate specification format for accomplishing the same result as the command GO TO XYZ (ijk). Refer to Figure 3 to use the following codes to specify gripper orientation. Alpha, beta, and gamma are in degrees.

Parameter 1	Parameter 2	Parameter 3
2.	alpha	beta
3.	beta	gamma
4.	gamma	alpha

The first parameter is the code indicating which two orientation angles will be specified in the next two parameters. The next two parameters are the orientation angles, in degrees. Three more parameters denote the location of the tool tip in terms of the robot coordinate system. The robot moves the gripper to the specified location and orientation. The roll angle will remain as before unless the ADJUST ROLL ANGLE command is simultaneously issued.

ADJUST ROLL ANGLE. There are two options here. For the first option, three parameters represent the roll vector -- the unit vector along the Z-axis of the gripper. If the vector is not known accurately, it may be approximated. If the approximation is inconsistent with other position specifications for the robot, the vector will be automatically adjusted, often to the actual roll desired. (The pertinent algorithm is explained in Bandy [2].) The reference for the roll vector is that it is parallel to the world Z-axis when the gripper is directed along the world X-axis and the roll angle is zero. An alternate specification may be used if the roll angle is actually known. In this case, the only parameter needed is the value of the roll angle in degrees.

ADJUST GRIPPER OPENING. The width between the fingers of the gripper which is mounted on the robot wrist adjusts to the dimension specified by a single parameter.

CHANGE GRIPPERS. A parameter in the principle period for this command specifies the gripper number: 1 for the parts handling gripper, or 2 for the tool handling gripper. This

first period must be followed by seven dummy periods. The robot moves the current gripper to its storage holster. The current gripper is released, and then the wrist goes to the holster containing the newly specified gripper, attaches itself to that gripper, and waits above the holster for the next command. Since the timing specification for this command does not follow the rule stated after the type definition of *period*, the requirement will be explained here. The timing specification for this command applies only to the robot moving the initial gripper to the approach point above the empty holster. The remaining gripper change actions take place during the eleven seconds after the initial gripper arrives above the empty holster. Therefore the start time for next robot command cannot be sooner than eleven seconds after the specified end time for the CHANGE GRIPPERS command.

4.4.2. Article-dependent Robot Commands

These commands are all related to handling articles. The coordinate system of an article is fixed to the article, and rotates and translates with respect to other systems as the article is moved. The *approach point* for an article is located at the tail of a designated approach vector of magnitude D when the head of the vector is at the centroid of the article. D is currently assigned the value 8.0 inches, but is easily changed. As with the Independent Robot Commands, when the robot moves its gripper from one location and orientation to another, the tool tip follows a linear trajectory at the same time that the attitude of the gripper also changes in a linear manner.

The robot places articles in World Locations. The codes for World Locations are to be used only in set files, and should not be confused with the Article Locations used in the file hartic.dat. The approach point of a World Location is calculated as if the article of current interest were situated in the location. Following are applicable World Locations:

100. Kardex Tray #1
101. Kardex Tray #2
102. Right Pallet Loading
103. Left Pallet Front Loading
104. Left Pallet Top Loading
105. Active Pedestal
106. Regrip Table
107. Tool Drum Loading

Some of the following commands refer to the *touch point* of an article. This point is the intersection of the line from the article centroid to the tool tip of the gripper, and the article surface nearest to the gripper.

GO APPROACH ARTICLE. The first parameter is the article number. The next three denote the approach vector, indicating the desired orientation of the roll axis of the robot, in the direction of the tool tip. Three more parameters represent the grip vector, indicating the direction of finger motion (which finger does not matter) when the robot grips the article. Both vectors are in terms of the coordinate system of the article. This first period must be followed by one dummy period. The robot moves the tool tip to the approach point of the specified article. At the same time, the gripper fingers open and the roll angle adjusts

according to the requirements for gripping the article. Note that when execution of this command is completed the gripper will still be a few inches away from the article.

GO GRIP. A parameter is equal to the article number, the only value needed in this command. This first period must be followed by one dummy period. The command GO APPROACH ARTICLE should have been issued prior to this command so that the gripper is at the approach point of the article. The gripper now moves to the article and then the fingers close. The specified end time should be one second greater than the time desired for the gripper to arrive at the article. The closing of the fingers occurs in the last second.

RETRACT WITH ARTICLE. All parameters are equal to zero. The command GO GRIP should have been issued prior to this command so that the gripper has closed on an article, but has not yet moved it. Now containing the article, the gripper moves back to the former approach point for the article.

GO APPROACH LOCATION. The first parameter is the World Location number. The next three denote the approach vector. Three more parameters represent the roll vector, directed along the local Z-axis of the gripper. Both vectors are in terms of the world coordinate system. This command may be issued when the robot gripper already contains an article. The robot carries the article to the approach point of the specified World Location. If the roll vector is not known accurately, it may be approximated. If the approximation is inconsistent with other position specifications for the robot, the vector will be automatically adjusted, often to the actual roll desired [2]. The reference for the roll vector is that it is parallel to the world Z-axis when the gripper is directed along the world X-axis and the roll angle is zero.

GO READY TO RELEASE. Parameters specify the World Location number and article number. The command GO APPROACH LOCATION should have been issued prior to this command so that the gripper is at the approach point of the World Location. The gripper now moves the article into its target place, but the fingers have not yet opened.

RELEASE / RETRACT. Parameters denote article number and a special code. This first period must be followed by one dummy period. The command GO READY TO RELEASE should have been issued prior to this command so that the article is in the desired place but the gripper has not yet released it. Now the fingers open, releasing the article, and the gripper backs off to the point indicated by the following interpretation of the code parameter. If the code equals zero, the gripper backs off to the approach point. If the code is not equal to zero, then the code is taken to be the desired distance between the fingertip and the touch point of the article. The duration between start and end times should be one second greater than the time it takes for the gripper to move from the article to the backoff point. The gripper does not start moving until after the first second, during which time the fingers open.

GO TOUCH. The article number is indicated by a parameter. This first period must be followed by one dummy period. This command is for moving the gripper from an initial position a short distance away from an article, to the point where the fingertips touch the article (i.e., to the touch point). The gripper should not initially contain an article.

TOUCH SPACING. Parameters denote article number and a special code. This first period must be followed by one dummy period. This command is for moving the gripper small

distances towards or away from an article, in the vicinity of the article. The gripper should not initially contain the article, so no release is done here; but otherwise this command is similar to RELEASE / RETRACT. The gripper moves to the point indicated by the following interpretation of the code parameter. If the code equals zero, the gripper moves to the approach point of the article. If the code is not equal to zero, then the code is taken to be the desired distance between the fingertip and the touch point of the article.

4.4.3. Vise Fixture / Pallet Commands

ADJUST JAW OPENING. A parameter is equal to the distance between the inner faces of the two jaws. The movable jaw of the vise fixture adjusts position to attain the specified distance between jaws.

CONVEY PALLET. The right pallet of the machine tool moves to the Y-coordinate specified by a parameter.

ROTATE PALLET. The right pallet of the machine tool rotates to the angle specified in degrees by a parameter.

4.4.4. V-block Fixture / Pallet Commands

ADJUST JAW OPENING. A parameter is equal to the distance between the inner faces of the two jaws. The movable jaw of the v-block fixture adjusts position to attain the specified distance between jaws.

CONVEY PALLET. The left pallet of the machine tool moves to the Y-coordinate specified by a parameter.

ROTATE PALLET. The left pallet of the machine tool rotates to the angle specified in degrees by a parameter.

4.4.5. Spindle / Tool Drum Commands

INDICATE MACHINING. All parameters are equal to zero. At the specified start time, the area about the spindle nose starts flashing yellow to symbolize machining. The specified end time has no effect on the duration of flashing, and should be assigned the same value as the start time. The flashing will last for about eight seconds. All actions in the workstation are suspended during that time, and consequently, subsequent commands should be timed as if machining lasted only an instant.

MOVE SPINDLE. Two parameters are equal to the X and Z coordinates, respectively. The spindle of the machine tool moves to the specified X (quill) and Z coordinates. The pertinent coordinate system originates at the spindle home position, and is parallel to the world coordinate system.

ROTATE TOOL DRUM. A parameter specifies the tool position number. The tool drum rotates until the tool position is situated for transferring a tool to the spindle.

4.4.6. Active Pedestal Commands

ADJUST JAW OPENING. A parameter specifies the distance between the two inner faces of the active pedestal jaws. The jaws move either towards or away from each other if necessary to attain the specified distance.

ROTATE. The active pedestal rotates to the angle specified in degrees by a parameter.

4.4.7. Kardex Commands

ARTICLE TO TRAY 1. The first parameter is the article number. The next three denote the world vector along the local X-axis of the article. The remaining three parameters represent the world vector along the local Z-axis of the article. The article will instantly appear on Kardex Tray 1 in the specified orientation. Prior to this command the article had to have been located in the Kardex hidden buffer.

ARTICLE TO TRAY 2. The parameters are exactly the same as for ARTICLE TO TRAY 1. The article will instantly appear on Kardex Tray 2 in the specified orientation. Prior to this command the article had to have been located in the Kardex hidden buffer.

4.5. Captions

For each active set file there must be an associated character string -- a caption to be shown at the bottom of the screen during animation as the set file is executing. *Active* set files are all sequential set files before the termination set file which contains only a dot (ASCII 46). The character strings must be contained in File Main/Data/Hdata/hcode.set. (See example in Figure 6.) There should be one character string per record, and each string must be enclosed in apostrophes. If captions are identical for some set files, then repeat them in separate records of hcode.set. If captions are not desired for some set files, then enter null character strings into the corresponding records of hcode.set. There must be at least as many character strings in hcode.set as there are active set files.

4.6. Set File Compression

As was mentioned before, a set file may contain as many as twenty periods (records). For purposes of developing and modifying the files, however, experience has shown that it is good practice to keep them short -- perhaps five to ten periods -- without being concerned that a large number of files may be generated this way. A subroutine named HCMBIN is provided with AWAP to combine all active set files into the smallest possible number of files before they are read. HCMBIN can either read files from some remote directory, leaving the files intact, and rewrite to the Hdata directory; or HCMBIN can both read and write from Hdata, writing over the first versions.

4.7. Set File Editor

When an animation sequence has been completed, the user has the option of repeating the same sequence, creating a modified sequence, or terminating the execution of AWAP. The Set File Editor allows the user to examine the initial conditions and animation data of the existing sequence, and to change any parameters as desired. The user may then have the *set file pointer* indicate which set file is to be executed first. The sequence may be run from that indicated file through the last file; or the indicated file may be executed alone.

VI. CONCLUSION

This document has described the *AMPLE* Version 0.1 prototype for HWS as it existed during the March 1987 AMRF benchmark test. Since that time, the *AMPLE* modules discussed above have been significantly improved, as will be explained in subsequent reports.

APPENDIX A:

AMPLE Communication Module (Acomm)

The *AMPLE* communication module is a means for transferring data and messages between the *AMPLE* system and suitably prepared AMRF workstations. During the March 1987 AMRF benchmark test, an **Acomm** link was demonstrated between *AMPLE* and both, HWS and the Turning Workstation (TWS). In this section, a description of the **Acomm** link to HWS will be presented. This link differs in some respects from the **Acomm** link to TWS [13].

Data sets transmitted between HWS and *AMPLE* include process plans in the flat-file format (Figure 1), various levels of control data for programmable controllers resident in HWS, and various error and status reports.

1. Acomm/HWS COMMUNICATION PROTOCOL

The communication protocol defines the requirements for the transmission of ASCII strings. Each string is terminated by a line feed (LF) character (ASCII 10). Every transmission begins with a *header record.* There are three kinds of header records, each identified by its initial keyword: ACCEPT-FILE, PROCESS, or SEND. The following pseudocode characterizes the organization of header records:

```
type HEADER is record
  keyword : 16 bytes;
  option : 16 bytes;
  file-option : 16 bytes;
  system-identifier : 16 bytes;
  data-identifier : 16 bytes;
  version-control : 16 bytes;
  LF-delimiter : 1 byte;
end record;
```

The ACCEPT-FILE header is used by HWS to prepare *AMPLE* to receive a named process plan in the flat-file format. The PROCESS header is used to signal *AMPLE* to begin the analysis of the named process plan in the *AMPLE* Process Planning Interface (APPI) module. The SEND header is used by HWS to request that the control data, prepared in response to the preceding header, be downloaded to the workstation. The control data to be transmitted are identified by file-option. If file-option is HWSC-CD, then error status reports are sent, followed by the control data for the HWS controller. If file-option is FIXT-CD, then control data are sent for the programmable fixture. Should an unrecognized file-option be requested, then **Acomm** transmits an ERROR response.

Acomm may make two responses to HWS: a DONE response which indicates the successful termination of a transmission, and an ERROR response which signals the presence of an error condition. The following pseudocode describes the organization of responses:

```
type RESPONSE is record
  byte-count : 10 bytes;
  keyword : 10 bytes;
  LF-delimiter : 1 byte;
  data : variable bytes;
end record;
```

The initial byte-count includes all LF delimiters in the variable-length data section of the response. If a response has no associated data, as is often the case, then the response consists of precisely 21 bytes.

2. Acomm OPERATION

The operation of **Acomm** is controlled by a program called **hwscomm**, which has been implemented in FranzLISP on a Silicon Graphics IRIS workstation. To establish a communication link between *AMPLE* and HWS, the command **hwscomm** should be entered, followed by a filename for the port through which the communication will take place. The following pseudocode describes the behavior of **hwscomm**:

```
procedure hwscomm (filename)

begin
  configure communication port;
  open communication port;
  open error-report;

loop
  receive header;

  get keyword from header;
  get planname from header;

  case keyword is

    when ACCEPT-FILE →
      send DONE response;
      print port to planname.pp;
      mark accept-file attribute of planname;
      send DONE response;

       when PROCESS →

      if accept-file attribute of planname is marked then
        initialize error-report and artci-report files;
        get process plan planname;
        get partname from planname;
        analyze planname;
```

```
    if analysis is successful then
      print artci-report to planname.out;
      start ARTCI process for planname and partname;
      mark process attribute of planname;

    else
      mark err-report attribute of planname;
      print error-report to planname.err;
      mark process attribute of planname;
    end if;

  end if;

send DONE response;

when SEND →
  if process attribute of planname is marked then

    get file-option from header;

    case file-option is

        when HWSC-CD →
      if err-report attribute of planname is marked then;
        print planname.err to port;
      else
        print planname.send to port;
      end if;

        when FIXT-CD →
        print planname.fixt to port;

        when others →
        file-option is unrecognized;

    end case;

  else
    planname was not PROCESS'ed;
    end ERROR response;

  end if;
```

```
            when others →
          keyword is unrecognized;
      end case;

  end loop

  end hwscomm;
```

Comments. This pseudocode version of **Acomm** omits many implementational details. To clarify the functional organization of this module, some of its operations will now be elaborated.

configure communication port;

The communication between *AMPLE* and HWS is established by an RS232C link to a port on the Silicon Graphics IRIS workstation. From the point of view of the IRIS workstation, HWS is treated as a certain kind of terminal. Using the UNIX **stty** command [20], the terminal options must be set to the following specification

-igncr:	ignore CR on input
-icrnl:	do not map CR to NL on input
-ixon:	enable START/STOP output control with ASCII DC1 and DC3
-ixoff:	request that system send START/STOP characters when input queue is nearly empty/full
-echo:	do not echo back every character typed
-echok:	do not echo NL after KILL character
-onlcr:	do not map NL to NL-CR on output

open communication port;

The filename passed as the only argument of the procedure **hwscomm** is stored and then used as a FranzLISP port name for the device configured in the preceding step. Because ports can be opened either in read-only or write-only modes, the implied bi-directional flow of messages and data in the communication protocol is actually implemented by repeatedly opening and closing the same port. Header records and responses are transmitted to ensure that all I/O operations are properly sequenced.

get *keyword* **from header;**
get *planname* **from header;**

Once a header record has been received, the information contained in it must be converted from a character string to internally usable data. The value of the symbol *keyword* is obtained by compacting the first 16 bytes of the header record. In this case, the compaction consists of stripping off all leading and trailing blanks. Embedded blanks are preserved, which means that ACCEPT-FILE is not the same keyword as ACCEPT FILE or ACCEPT -FILE. The

value of the symbol *planname* is obtained by completely compacting the data-identifier and version-control fields of the header record. Not only are all leading and trailing blanks removed, but all embedded blanks are also removed.

print port to *planname*.**pp**;

To prevent the loss of data when an ACCEPT-FILE keyword is received from HWS, **Acomm** initiates a background process which causes data from the communication port to be transferred to a file. Specifically, the terminal device which has been configured as a communication port is taken as the redirected input to the UNIX **cat** function and the file *planname*.**pp** is taken as the output. The **cat** operation is terminated, and the output file is closed, when an EOT character is received.

mark *accept-file* **attribute of** *planname;*
mark *err-report* **attribute of** *planname;*
mark *process* **attribute of** *planname;*

In **Acomm** the processing stages of a specific *planname* are indicated by *marking* the indicated attributes of the symbol *planname*. Appendix B explains how such marking of symbol attributes is done in FranzLISP, the language in which **Acomm** is implemented.

get process plan *planname;*
analyze *planname;*

These operations are provided by the *AMPLE* Process Planning Interface, which was discussed in Section III.

start ARTCI process for *planname* **and** *partname;*

The ARTCI process and the construction of the files *planname*.**send**, *planname*.**fixt**, and *planname*.**err** are discussed in Section IV.

APPENDIX B:

LISP-Related Notes on **Acomm** and APPI

The software discussed in this report took experimental advantage of concepts which will be central to subsequent versions of the *AMPLE* system. While a simplified and specific approach to building internal representations satisfied the needs of this application, future implementations will accomplish such tasks by means of a general, comprehensive tool: the *AMPLE* Object Oriented Editor (OED) [7]. **Acomm** and APPI, as well as the central components of the *AMPLE* system -- *AMPLE/core*, OED and others (not mentioned in this report; see *"AMPLE/mod"* in Glossary) -- are written in FranzLISP.

One advantage of LISP as an implementation language is that it permits a very precise control over the name space of a particular application. Symbols may be assigned values, may be assigned function definitions, and may have an extensive list of attributes or properties. In FranzLISP, control over the property list is achieved with the functions **putprop** and **get**. That is, the expression

(putprop 'my-tree 'red 'leaf-color)

says that the **leaf-color** attribute of **my-tree** is **red**. The function **get** returns the value of the attribute. That is, if the expression

(get 'my-tree 'leaf-color)

were evaluated, then it would return the value **red**. If the symbol does not have the attribute in question, then the value returned is the empty list object, called **nil**. In Boolean evaluations, **nil** is equivalent to the Boolean **false**. An example of how attributes have been used is seen in **Acomm**, where the processing stages of a specific *planname* are indicated by *marking* the indicated attributes of the symbol *planname;* that is, by assigning the attribute some non-**nil** value. Since any such value would do, the specific value assigned is **t**, which is the equivalent of the Boolean value **true**.

The process plan format has also been designed to take advantage of LISP. In flat file process plans, the legality of keywords and other symbols and values reflects the conventions of the underlying LISP readtable. In order to resolve process plans into tokens, this readtable has to be revised. In FranzLISP, colons, semicolons, and parentheses are special symbols. Colons are used to describe access paths in packages, semicolons to initiate comments, and parentheses to initiate (and terminate) lists. Thus, before any process plan is parsed, the standard FranzLISP readtable is modified by reclassifying these symbols to the symbol class **vcharacter** using the FranzLISP function **setsyntax**. The **initialize** *readtable* operation changes the status of certain tokens in this manner as the first step of the first pass analysis of APPI.

A very important function of *AMPLE/core* is to keep a detailed map of the name space, which is a net of externally visible names and their semantic relations with one another. Nets of this kind may be called *symboltables*. When one is analyzing process plans for HWS, the symboltable would contain such information as:

An *AMPLE* Prototype: HWS

LOAD **isa** *workelement* **of** HWS

FIXTURE-ID **isa** *attribute* **of** LOAD

type FIXTURE-ID **is string**

The ease with which information of this kind can be represented in LISP is one of the most persuasive reasons for using this programming language in this implementation.

LIST OF REFERENCES

[1] Albus, J.S.; Barbera, A.J.; and Nagel, R.N. "Theory and Practice of Hierarchical Control," *Proceedings of the 23rd IEEE Computer Society International Conference*, 1981 September. 18-39.

[2] Bandy, H.T. *The AWAP HWS Prototype*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[3] Bandy, H.T. *The AWAP System Specification*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[4] Bandy, H.T.; Parker, J.S.; and Carew, V.E., Jr. *The AMPLE Coordinate Editor*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[5] Boudreaux, J.C. AMPLE: A Programming Language Environment for Automated Manufacturing. Boudreaux, J.C.; Hamill, B.; and Jernigan, R., editors. *The Role of Language in Problem Solving - 2;* North Holland, Amsterdam; 1986.

[6] Boudreaux, J.C. *The AMPLE Project*, National Bureau of Standards (U.S.) NBSIR 86-3496; 1987 March. 13 pages.

[7] Boudreaux, J.C. *OED: The Object-Oriented Editor*, National Bureau of Standards (U.S.) NBSIR 87-3530; 1987 March. 15 pages.

[8] Brown, P.F.; and Ray, S.R. *The NBS-AMRF Process Planning System: System Architecture*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[9] Bunch, W.R. *The Material Buffering System of the Horizontal Workstation*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[10] Chasen, S.H. *Geometric Principles and Procedures for Computer Graphic Applications*, Englewood Cliffs, NJ: Prentice-Hall; 1978. 241 pages.

[11] Fishman, D. *The High Level Machine Tool Controller of the Horizontal Workstation*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[12] *IRIS User's Guide Volume I (Version 3.0)*, Mountain View CA: Silicon Graphics, Inc.; 1986.

[13] Lee, K.; Donmez, A.; Gavin, R.; Greenspan, L.; Lee, V.; Reisenauer, E.; Peris, J.P.; Shoemaker, C.; and Yang, C. *The Turning Workstation in the AMRF*, National Bureau of Standards (U.S.) NBSIR 88-3749; 1988 March. 197 pages.

[14] Nashman, M. and Chaconas, K.J. *The NBS Vision System in the AMRF*, National Bureau of Standards (U.S.) NBSIR 87-3684; 1987 December. 42 pages.

[15] Paul, R.P. *Robot Manipulators: Mathematics, Programming and Control*, Cambridge MA: The MIT Press; 1981. 279 pages.

[16] Scott, H.A. *Architecture and Principles of the Horizontal Workstation*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[17] Scott, H.A. and Burton, R.J. *The Automated Fixturing System of the Horizontal Workstation*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[18] Simpson, J.A.; Hocken, R.J.; and Albus, J.S. "The Automated Manufacturing Research Facility," *Society of Manufacturing Engineers Journal of Manufacturing Engineering*, 1(1): 17-32; 1982.

[19] Strouse, K. and Scott, H.A. *Horizontal Workstation Controller Implementation*, to be published as a National Bureau of Standards (U.S.) NBSIR.

[20] *UNIX Programmer's Manual Volume 1A*, Mountain View CA: Silicon Graphics, Inc.; 1986.

[21] Wavering, A.J. and Fiala, J.C. *The Real-Time Control System of the Horizontal Workstation*, National Bureau of Standards (U.S.) NBSIR 87-3692; 1987 December. 183 pages.

GLOSSARY

active set files -- In AWAP, all sequential set files before the termination set file. Active set files contain data that determine motions in an animation sequence.

AMPLE -- The Automated Manufacturing Programming Language Environment. *AMPLE* provides a mechanism for constructing control interfaces to industrial manufacturing processes, and also provides an integrated system of software tools for translating product design and process planning specifications into verified equipment-level control programs.

AMPLE/core -- The central processor of *AMPLE*, which contains appropriate representations of all entities in the manufacturing domain.

AMPLE/mod -- A loosely bundled, extensible collection of software modules distributed around *AMPLE/core*. Some modules, such as the Workspace Manager (WM), the Lexical Analyzer (Lexx), and the Object Oriented Editor (OED), permit direct access to *AMPLE/core*. Other modules are less tightly coupled to *AMPLE/core*, and provide the programmer with specific support functions.

AMRF -- The Automated Manufacturing Research Facility at the National Bureau of Standards.

animation clock -- In AWAP, the concept regulating the timing of animated motions. Seconds in time are either specified or implied in set file commands. The resulting animation timing depends on the screen refresh rate, and is accurate unless calculations between frames slow the programmed refresh rate of 7.5 frames per second. All motions cease during the eight seconds that the INDICATE MACHINING command takes to execute, and then motions resume their timing as if there were no pause. This means that the "animation clock" was suspended during that pause, and the execution of the set file will actually take eight extra seconds.

animation sequence -- The collective result of the successive execution of all active set files.

APPI -- The *AMPLE* Process Planning Interface. This module reads, parses, and verifies process plans in the AMRF flat-file format.

approach point -- (of an article:) The point located at the tail of a designated approach vector when the head of the vector is at the centroid of the article to be approached. In AWAP, the magnitude of an approach vector is set to always be 8.0; however the value of this constant may easily be reassigned. The approach point of a World Location is calculated as if the article of current interest were situated in the location.

approach vector -- The unit vector in the direction in which the robot gripper is to approach an article from a nearby location. When the approach point is reached, the roll axis will be aligned with the approach vector.

ARTCI -- The *AMPLE* Real Time Control Interface. This module generates and verifies control data for manufacturing workstations.

Article Location -- The code used in AWAP for specifying the location of an article.

article -- A tool, workpiece or any other item handled by the robot.

ASCII -- American Standard Code for Information Interchange, a standard form in which computer data may be stored and transmitted.

AWAP -- The *AMPLE* Workstation Animation Package. This module produces an animated simulation of the workstation in action.

background process -- A computer process that is not directly accessible to the user. Such processes are frequently used to perform maintenance and updating on computer systems.

caption -- A character string shown at the bottom of the screen during animation, corresponding to a set file.

component -- Also called "major component" -- In AWAP, a grouping of "members" which has its own coordinate system. Examples of components are the robot, the v-block fixture, etc.

directed graph -- Also called "digraph" -- A finite but unempty set of vertices or nodes, which are connected by a (possibly empty) set of ordered pairs of vertices whose members are called directed edges, or arcs.

end time -- In AWAP: except for special cases noted in this report, the time for the motion for the pertinent period to stop. End time is specified in seconds, measured from the start of the current set file.

fingertip -- The extreme end of a finger of a robot gripper. In AWAP, since the length of a finger is along the X direction of the finger coordinate system, the fingertip has a greater X coordinate than any other part of the finger. For the current model of the HWS robot, the finger system X coordinate of the tool tip is 1/4 inch less than the finger system X coordinate of the fingertip.

finite state machine -- A system whose next function is determined according to a data table. The table consists of multiple rows and columns. Each column represents an input or output value for the system, and each row represents an event or set of events. The system scans the table for a row containing a particular set of input values. When such a row is found, the finite state machine will output all of the values found in the output columns of that row.

fixture -- A mechanical system for rigidly attaching workpieces to a machine tool table. The fixtures of HWS grasp and release workpieces under computer control.

flat file format -- The AMRF standard format for storing process plans.

gripper orientation -- In AWAP for the HWS robot, the orientation of the roll axis, which is also the local X-axis, of the gripper. This specification is without regard to the roll angle, which is specified independently. Gripper orientation may be specified in terms of either a unit vector or the alpha, beta and gamma codes depicted in Figure 3.

grip pose -- A definite manner in which a robot is to grasp a specific part. The grip pose describes the gripper contact points on the part, along with the orientation vector of the gripper as it approaches the part.

grip vector -- A unit vector indicating the direction of motion of either of the robot gripper fingers when an article is to be gripped. The vector for either finger has the same effect.

group -- Also called "motion group" -- In AWAP, a collection of geometric entities whose changes in position depend on a common set of equations. In some cases a group is made up of members whose motion with respect to one another is determined by a single command.

HLMC -- The High-Level Machine tool Controller. This is a computer system that was added to the controller provided with the HWS horizontal machining center so that the machining center could perform additional functions.

horizontal machining center -- A milling machine used for metal and plastics cutting. A machining *center* implies certain features such as the ability to store multiple tools and load them under computer control. A horizontal machining center has its spindle oriented in the horizontal plane.

hsnarc -- The name of the executable image for the HWS edition of AWAP. The UNIX shell script Main/Hws/hsnarc executes the actual executable image Main/Execut/hsnarc.

HWFC -- Horizontal Workstation Fixturing Controller. This computer system controls all of the fixtures in HWS.

HWS -- The Horizontal Workstation of the AMRF at the National Bureau of Standards. HWS consists of a robot, a horizontal machining center and other components. The HWS edition of AWAP provides an animated simulation of HWS.

HWSC -- Horizontal Workstation Controller. This computer system controls HWS. It schedules events and coordinates activity between systems in the workstation.

lower axis (of coordinate plane:) -- The first of the two axes identifying a coordinate plane according to right hand rotational order (X, Y, Z, X, Y, etc.). For example, the lower axis of the Y-Z plane is Y; the lower axis of the X-Z plane is Z.

MacRow -- Term used to describe a macro used by the HWSC. Derived from computer MACro and a ROW of a state table.

MBC -- Material Buffering Controller. The computer system that controls the material buffering device in HWS.

NC -- Numerical Control -- The term referring to the technology used to automate machine tools. Numerically controlled machine tools are programmable to operate under computer control.

parse -- The process of resolving a string of symbols, known to be a grammatical sentence of a given language, into its syntactic components.

period -- In AWAP, a time interval in which action for a specified "motion group" takes place. Each record in a set file represents a "period". Periods are referred to as the sequence numbers of the records in the set file.

permanent data (for articles:) -- Dimensional constants for articles, stored in the odd-numbered records of in hartic.dat and used for seating articles in Article Locations.

port -- Also called "communication port" -- A connection point to a computer system.

pseudocode -- A form of notation similar to a computer language, used for program design or description.

RCS -- Real-time Control System. The RCS controls the robot in HWS.

robot system -- In AWAP for the HWS robot: the coordinate system of the robot, which is parallel to the world coordinate system, but whose origin is located at World [0., 0., 59.42].

roll axis -- In AWAP for the HWS robot, the local X-axis of the gripper. The +X direction is used as a specification for gripper orientation. (See Figure 2.)

roll vector -- In AWAP for the HWS robot, a unit vector along the local Z-axis of the gripper (Figure 2). The direction of the roll vector implies the roll angle of the gripper -- i.e., how far the gripper is rotated about its X-axis from its reference position.

RS232C -- A widely used standard electrical interface for interconnection of computer equipment.

seating surface -- A surface of the article, which may contact some surface of a location in which the article may be placed.

set file -- A data file that determines the motions for some segment of the duration of an AWAP animation sequence. The only exception to this definition is the last set file of the animation sequence, called the "termination set file", which contains only a dot.

start time -- In AWAP, the time for the motion for the pertinent period to start. Start time is specified in seconds, measured from the start of the current set file.

state table -- A table of data used to operate a system defined as a finite state machine. In a state table each row of data represents a possible state of the system, and the action to be taken should the system be in that state.

symbol table -- A list, kept by a language translator of a system, of words and symbols known to the system.

template -- A pre-defined character string having blanks which are to be replaced with values to be assigned by the software making use of the template.

termination set file -- The file following the last active set file. The termination set file contains only a dot.

timeline -- A representation of the state of a system at a specified time, in terms of the status of each element of the system at that time.

token -- A syntactically significant unit (keyword, name, etc.) or sequence that is an input to a software system.

tool tip (of the robot gripper:) -- The reference point fixed in the local coordinate system of the gripper, which is meant to coincide with target world space coordinates of a gripper location specification.

touch point (of an article:) -- The intersection of the line from the article centroid to the tool tip of the robot gripper, and the article surface nearest to the gripper.

tray sector -- In the AMRF, a region within a material handling tray. The AMRF data base keeps track of which part or tool is in each sector of each tray, and this information may be used by the robot, the vision system, or the material handling system.

vision system -- A optical system used by robots and other automated devices for the purpose of identifying objects and ascertaining their locations and orientations. This information is used by the HWS robot to identify and acquire parts that are stored in material handling trays.

world coordinate system (for the HWS edition of AWAP:) -- The Cartesian coordinate system which originates at the center of the base of the robot (on the floor), has the X-axis pointed toward the machine tool, and has the Z-axis pointed upward.

World Locations -- AWAP codes indicating where the robot is to place articles. The codes for World Locations are to be used only in set files, and should not be confused with the Article Locations used in the file hartic.dat.

world space -- 3-dimensional space in terms of the world coordinate system.

INDEX

READER COMMENT FORM

Document: **An AMPLE Version 0.1 Prototype: The HWS Implementation**

This document is one in a series of publications which document research done at the National Bureau of Standards Automated Manufacturing Research Facility from 1981 through March, 1987.

You may use this form to comment on the technical content or organization of this document or to contribute suggested editorial changes.

Comments: _____

_____

_____

_____

_____

_____

_____

_____

If you wish a reply, give your name, company, and complete mailing address: _____

_____

_____

_____

What is your occupation? _____

NOTE: This form may not be used to order additional copies of this document or other documents in the series. Copies of AMRF documents are available from NTIS.

Please mail your comments to:

AMRF Program Manager
National Bureau of Standards
Building 220, Room B-111
Gaithersburg, MD 20899

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR88-3770	2. Performing Organ. Report No.	3. Publication Date APRIL 1988

4. TITLE AND SUBTITLE

An AMPLE Version 0.1 Prototype: The HWS Implementation

5. AUTHOR(S)

Herbert T. Bandy, Victor E. Carew, Jr., Jack C. Boudreaux

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The Automated Manufacturing Programming Language Environment (AMPLE) system is being developed within the Center for Manufacturing Engineering of the National Bureau of Standards to provide a uniform programming language environment for the construction of control interfaces to industrial manufacturing processes; and to provide an integrated system of software tools for translating product design and process planning specifications into equipment-level control programs. Work on the AMPLE project has been surrounded by a larger and more comprehensive project which investigates the design of advanced automated manufacturing systems. This project, embodied in the Automated Manufacturing Research Facility (AMRF) of the National Bureau of Standards, has provided an invaluable source of empirical data and practical experience. In this report the modules of the implementation of the AMPLE Version 0.1 prototype for the Horizontal Workstation System (HWS) will be described.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Automated manufacturing; computer animation ; programming language environment; rapid prototype; real-time control.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

77

15. Price

$13.95

USCOMM-DC 6043-P80