SRC TR 87-97

TECHNICAL RESEARCH REPORT

An Integration of Manufacturing Resource Planning (MRP II) and Computer Aided Design (CAD) Based on Update Dependencies

by

G. Harhalakis, L. Mark, M. Bohse, and B. Cochrane

SYSTEMS RESEARCH CENTER

UNIVERSITY OF MARYLAND

COLLEGE PARK, MARYLAND 20742

An Integration of Manufacturing Resource Planning (MRP II) and Computer Aided Design (CAD) Based on Update Dependencies

G. Harhalakis[1], L. Mark[2], M. Bohse[1], B. Cochrane[2]

[1]University of Maryland, Department of Mechanical Engineering
[2]University of Maryland, Department of Computer Science

ABSTRACT. The traditional, fragmented approach to increasing manufacturing efficiency has resulted in "islands of automation" in our factories. Computer Integrated Manufacturing (CIM) is the goal of tying together these islands into a single coherent system capable of controlling an entire manufacturing operation. The technical and organizational difficulties of such a massive undertaking require a modular approach to CIM implementation, with an initial nucleus being gradually expanded by allowing interaction between it and other systems' databases. Manufacturing Resource Planning (MRP II) is best positioned to serve as this nucleus. The suggested first system for integration is Computer Aided Design (CAD); the integration being centered around part specification, product structure, and engineering changes. A model of the CAD/MRP II integrated system, detailing the logical interaction between the systems in the areas of part specification maintenance and engineering changes, is currently being developed. A formal language for specifying the operations in and between the MRP II and CAD systems, namely "update dependencies," has been defined, and used as an AI production system. In addition, an interpreter for the specification language has been implemented in Prolog and has been tested against a portion of the model design specifications. A demonstration session is also presented with results and a discussion of our experience so far. Finally, the next steps of our implementation strategy are outlined.

INTRODUCTION. Under pressure to remain efficient and competitive, many companies feel compelled to implement one or more of the vast array of new technologies and techniques which are being presented and promoted as a means to the development of the factory of the future. These include Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Flexible Manufacturing Systems (FMS), Manufacturing Resource Planning (MRP II), Group Technology (GT), Just In Time Inventory Control (JIT), Automated Materials Handling (AMH) and Computer Aided Process Planning (CAPP), to name only a few. Too often, however, this approach, in which individual technologies are implemented independently, results in "islands of automation", where individual tasks are automated without any communication or interfacing with other related activities.

Instead of firms independently automating as many as 50 different functional areas [1] often using unique hardware and software for each, it is time to adopt a systematic approach to implementing and integrating the various technologies as a means for achieving the productivity gains required [2].

In order to coordinate all of these activities, MRP II may effectively serve as the "hub" of the CIM system, [3] as depicted in Figure 1, the pro-

posed functional model of CIM. The links between the various systems are determined by the common information required and the logical rules to regulate the data flow.

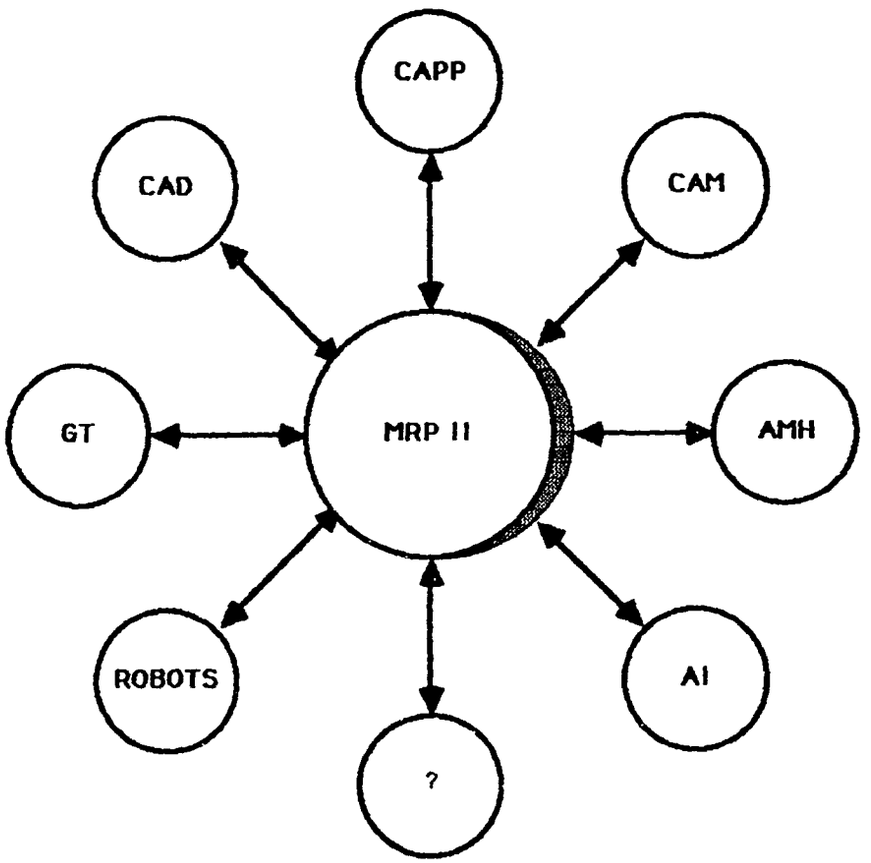

Fig. 1 Functional CIM Model with MRPII as the "Hub"

There are many problems and issues that must be resolved before CIM is possible, among which the database architecture is of utmost importance [4]. Given the functional model of CIM proposed in Figure 1, how is the actual system to be constructed? There are two primary schools of thought in this area. The first is that a single database, accessible to all system functions and maintaining all system data, should be constructed. The second alternative is that separate databases be maintained for each function, and interoperational capabilities be added as needed. We support and furthermore endorse, the separate database solution which facilitates a gradual evolution towards CIM, and carries the promise of software vendor independence. We feel that the separate database solution carries sufficient promise to warrant further investigation. As seen later, multi-database interoperability depends on artificial intelligence, in the form of rule-based expert systems, in order to define the proper interaction between the databases involved, under all circumstances.

This paper discusses a suggested model for the functional integration of CAD and the Bill of Material module of an MRP II system, given the fundamental

similarities of operations and commonality of data between these systems. The functional design and the detailed description of the model are followed by the first steps toward the implementation of it, using the multi-database interoperability technique discussed in later sections. The next steps required to reach a state of functionality follow the conclusions drawn from this first part of the work.

THE FUNCTIONAL DESIGN OF THE MODEL. The systems under consideration for this starting point of our CIM model are Computer Aided Design (CAD) and Manufacturing Resource Planning (MRP II). While neither of these can be called fully mature, their overlap is well established: product definition. CAD facilitates the creation and design of parts and assemblies, where assemblies are really just arrangements of component parts. MRP II has the role of cataloging each part and assembly by number and description and defining the product structure (i.e. where each part is used).

More specifically, the elements common to MRP II and CAD addressed by the integration are as follows:

- Part Specifications
- Bills of Material
- Engineering Changes

The functional model of the CAD/MRP II integrated system is based on the similarity of functions and the commonality of data between the two systems. The model is not derived from any two commerical packages in particular, but instead is intended to be generic enough to be applied to any set of fairly well-designed systems. The model to be presented includes the sharing of part specification and engineering change data. The model is intended to operate in a discrete parts, make-to-stock environment.

The part specification data maintained by each system is shown in Figure 2. General part data is maintained for each part and is retrieved by part number; in addition to this data, the effectivity start and end dates and status code (different for each system) of each revision is maintained.

For Each Part Number

CAD	MRP II
Part Number	Part Number
Drawing Number	Drawing Number
Drawing Size	Drawing Size
Description	Description
CAD Unit of Measure	CAD(BOM) Unit of Measure
	MRP(Purchasing) Unit of Measure
	UOM Conversion Factor
	Source Code
	Cost
	Leadtime
Supersedes Part Number	Supersedes Part Number
Superseded by Part Number	Superseded by Part Number

For Each Revision Level

CAD	MRP II
Part Number	Part Number
Revision Level	Revision Level
Effectivity Start Date	Effectivity Start Date
Effectivity End Date	Effectivity End Date
CAD Status Code	MRP II Status Code

Fig. 2 Part Specification Data Maintained by Each System

It is assumed that no data exists in either system when the integration is established, ensuring data consistency.

The functioning of the model can be represented by examining the status codes associated with each part and revision. These codes have different values for each system, as follows:

CAD Status

W - "Working": not a completed drawing, used prior to approval, and not transmittable to MRP II

R - "Released": an active part

H - "Hold": under review, pending for approval, possibly with a new revision level. Part should not be used by either system.

O - "Obsolete"

MRP II Status

R - "Released": active part

H - "Hold": not to be used by MRP

The basic functions of the system are described with the aid of status code diagrams, showing the flow of information and the status of each part in both systems during a given activity. In the following sections, the basic operations are described through the presentation of appropriate scenarios.

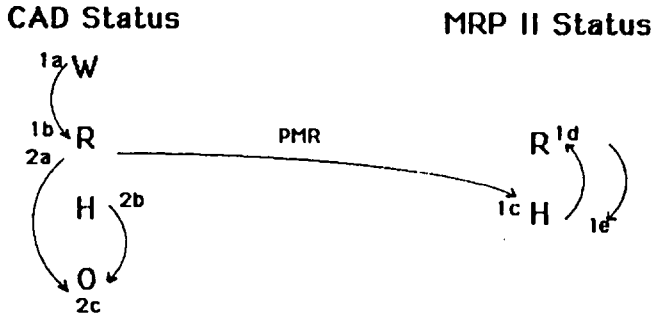

Fig. 3 Status Diagram for the Creation of a New Part

A brand new part is first created by a CAD user as a working drawing (figure 3, point 1a). At this point, no information about the part exists in MRP II. Upon completion and approval within CAD, the part is released by a CAD user (1b).

If the part supersedes another, the status of the superseded part is immediately changed in CAD to obsolete, (2c), regardless of whether the part previously had an R status (2a) or an H status (2b). In MRP II, the changeover to the superseding part is performed automatically, by virtue of the effectivity start date of the higher level assembly calling for the new part as part of a revision change, handled by an Engineering Change procedure.

The release of the new part within CAD triggers the establishment of a skeletal Part Master Record (PMR) in MRP using the CAD Part Specification data. Because the PMR is not complete, and to give manufacturing time to plan for the purchase or manufacture of the part (eg., search for vendors, develop routings) the part is given a status of H in MRP II (c). When MRP II users complete the PMR, the part can be released within MRP II (d). If the need arises, due to a machine break down or vendor problems, for example, MRP II users can place a local hold on the part (e) without affecting CAD. Once held, MRP II users can again release the part.

Similar diagrams have been developed for part obsolescence, deletion, and changes of revision code.

The basic interopeability functions required to maintain consistent assembly, or Bill of Material, information, have also been developed. These address the addition of components to assemblies, the deletion of components from assemblies, and the replacement of one component with another.

DATABASE INTEROPERABILITY. The update dependency formalism has a wide variety of applications in walk-through guidance control systems, cause-effect systems, statistical information gathering, knowledge acquisition, policy enforcement, and production control.

Within the database area we are currently applying update dependencies to specifying and controlling aspects of databases ranging over integrity constraints, transactions, normalization, view maintenance and update, and metadata management.

In this section we define the syntax and the semantics of update dependencies.

Syntax

A compound update operation is defined by an update dependency with the following form:

```
<op>
  ->   <c1>,
       <op1,1>,
       <op1,2>,
       ..
       <op1,n1>.
  ->   <c2>,
       <op2,1>,
       <op2,2>,
       ..
       <op2,n2>.
  ->   ..
```

where <op> is the compound update operation being defined, <opij>, is either an implied compound update operation or an implied primitive operation, and <ci> is a condition on the database state.

A compound update operation <opi> has the following forms:

- <operation_name>(<relation_name>(<tuple_spec>))

where the <tuple spec> is a tuple variable for the relation with the name <relation name> and consists of a list of <domain variable>s. The <tuple spec> in <op> is the formal parameter for <op>. All the <domain variable>s in the <tuple spec> of <op> are assumed to be universally quantified. All <domain variable>s in the <tuple spec>s of <opij>, that are not bound to a universally quantified <domain variable> in <op>, are assumed to be existentially quantified. All <domain variable>s are in caps; nothing else is.

The implied primitive operators are: 'add' for adding a new tuple in a relation, 'remove' for eliminating one, 'write' and 'read' for retrieving data from the user, 'new' for creating a unique new surrogate, and 'break' for temporarily stopping the system to do some retrievel before giving the

control back to the system. The implied primitive operations <opij> have the following forms:

- add(<relation name>(<tuple spec>))
- remove(<relation name>(<tuple spec>))
- write('<any text>'), or write(<domain variable>)
- read(<domain variable>)
- new(<relation name>(<tuple spec>))
- break

The <relation name> used in the operation 'new' must be the name of a unary relation defined over a non-lexical domain. The conditions <cond> are expressions of predicates. The conectives used in forming the expressions are 'and' and 'not'. The predicates are of the form <relation name>(<tuple spec>) to determine whether or not a given tuple is in a given relation; or of the form 'nonvar(X)' or 'var(X) to decide whether or not a <domain variable>, X, has been instantiated; or of the form X<comp>Y, where <comp> is a comparison operator.

Conditions, or retrieval dependencies, can also be used to retrieve data from the system.

Semantics

A compound update operation succeeds if, for at least one of the alternatives in the its update dependency, the condition evaluates to true and all the implied operations succeed. It fails otherwise.

When a compound update operation is invoked its formal parameters are bound to the actual parameters. The scope of a variable is one update dependency. Existantially quantified variables are bound to values selected by the database system or to values supplied by the interacting user on request from the database system. Evaluation of conditions, replacement of implied compound update operations, and execution of implied primitive operations is left-to-right and depth-first for each invoked update dependency. For the evaluation of conditions we assume a closed world interpretation.

The non-deterministic choice of a replacement for an implied compound update operation is done by backtracking, selecting in order of appearance the update dependencies with matching left-hand sides. If no match is found, the operation fails.

An implied compound update operation matches the left-hand side of an update dependency if:

- the operation names are the same, and
- the relation names are the same, and
- all the domain components match. Domain components match if they are the same constant or if one or both of them is a variable. If a variable matches a constant it is instantiated to that value. If two variables match they share value.

The semantics of the primitive operations are:

- add(r(t)); its effect is r := r U {t}; it always succeeds; all components of 't' are constants.
- remove(r(t)); its effect is r := r\{t} where all components of 't' are constants. It always succeeds.
- write('text'); it writes the 'text' on the user's screen. It always succeeds.
- write(X); writes the value of 'X' on the user's screen. It always succeeds.
- read(X); reads the value supplied by the user and binds it to 'X'. It always succeeds (if the user answers).
- new(r(D)); produces a new unique surrogate, from the non-lexical domain over which 'r' is defined and binds the value of the variable 'D' to this surrogate. It always succeeds.
- break; suspends the current execution and makes a new copy of the interpreter available to the user, who can use it to retrieve the information he needs to answer a question from an operation.

The list of primitive operations is minimal for illustrating the concept. It can easily be extended. It is emphasized that primitive operations are not available to the user; he cannot directly invoke them.

The execution of 'add' and 'remove' operations done by the system in an attempt to make a compound update operation succeed, will be undone in reverse order during backtracking. This implies, that a (user invoked) compound update operation that fails will leave the database unchanged.

We have recently found that we can automatically generate update dependencies for a number of high level control abstractions, including while, repeat, do-for, if-then-else, and case statements. We therefore expect to revise the language to include these abstractions, and thus become more user-friendly.

Using the Update Dependency Language, operations can be coded to implement the functional design of the integrated MRP II/CAD system. In Appendix A, the code of five somewhat simplified operations used to insert new parts and revision changes from either MRP II or CAD into the integrated system are presented. These operations are specified such that the system prompts the user for any required input fields not specified by the user in the call to the operation. Calls to the operations do not succeed unless all of the calls to other operations within the operation succeed, assuring the consistency and integrity of the two databases.

DEMONSTRATION. A sample interactive session demonstrating some of the basic interoperability functions of the MRP II/CAD system is shown in Figures 4-7. Initially, both the MRP II and CAD databases are empty. The CAD and MRP II part and revision records contain the information specified in Figure 2. In Figure 4, a new part is inserted into the CAD database; the user is prompted for the data required to establish a record for the new part and one for its

```
d> insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)).

art Number?
: 12345.
escription?
: deluxe_widget.
nit of Measure?
: each.
ew Revision Level?
: 1.
rawing File Name?
: 'widget.prt'.
evision Has Been Added
art Has Been Added


d> listing(cadpart).

adpart(12345,unknown,unknown,deluxe_widget,each,unknown,unknown).

d> listing(cadrevs).

adrevs(12345,1,unknown,unknown,w,'widget.prt').

d> listing(mrppmr).

d> listing(mrprevs).
```

igure 4. Adding a New Part to CAD.

```
d> releasework(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

art Number?
: 12345.
evision Level?
: 1.

evision Has Been Released

d> listing(cadrevs).

adrevs(12345,1,unknown,unknown,r,'widget.prt').

d> listing(mrppmr).

rppmr(12345,unknown,unknown,deluxe_widget,each,unknown,unknown,unknown,
      unknown,unknown,unknown,unknown).

d> listing(mrprevs).

rprevs(12345,1,unknown,unknown,h).
```

igure 5. Releasing a New Part

```
 insert(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

t Number?
12345.
 Revision Level?
2.
wing File Name?
'widget2.prt'.

ision Has Been Added

 listing(cadrevs).

revs(12345,1,unknown,unknown,r,'widget.prt').
revs(12345,2,unknown,unknown,w,'widget2.prt').

 listing(mrprevs).

revs(12345,1,unknown,unknown,h).
```

ure 6. Adding a New Revision to CAD.

```
 insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)).
t Number?
12345.
cription?
hammer.
t of Measure?
each.

t Number Already Exists

 releasework(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).
t Number?
12345.
ision Level?
1.

t Does Not Have Working Status
```

ure 7. Preventing Users from Compromising Database Integrity.

first revision level. The remaining fields, if not specified by the user in the call to the operation, are given the value "unknown," as shown in the subsequent listing of the CAD part and revision records. The absensce of the MRP II part master and revision records verifies that no information has been transferred to MRP II yet.

In Figure 5, the first revision of the part just created is released. The listing of records after the part's release shows the updating that has occurred in both CAD and MRP II. In CAD, the status of the revision data is changed from working to released; in MRP II, a part master record is created for the new part, as well as a revision record with a status of "hold" for the first revision of the part. Any data not included in the CAD record is given the value "unknown".

Figure 6 shows the addition of a new revision level to the part. The listings after this addition show the revision record with a working status in CAD, but not recorded at all in MRP II.

The system is designed to prevent the user from inadvertently compromising the integrity and consistency of the two databases. Figure 7 shows two examples of this. In the first example, the user attempts to enter a new part using a part number that already exists. The system prints a message out indicating this and stops the operation. In the second example, a user tries to release a CAD revision that is already released. Again the system prevents this action and prints a message.

DISCUSSION. The sample operations performed in the previous section represent only a small portion of the interoperability system, which extends to many other typical part maintenance functions. Even so, the model is a simplification of the activities and data exchange involved in a typical organization, a result of the desire to remain as general as possible in the solution to MRP II/CAD integration. In its present version, the usefullness of the model is limited by its isolation from actual CAD and MRP II systems, forcing the users to interact directly with the interoperability system. As the research progresses, the sophistication of the model will increase and the interoperability system will become more transparent to the user, interacting with the application programs directly, although the latter seems to be a remote goal at this point in time.

Update dependencies provide a convenient formalism for organizing, expressing, and communicating algorithms. Update dependencies have a declarative representation tht allows the designer to express algorithms in a natural manner. This natrual representation is appropriate for reviews and presentation since it is a consise statement of the user'snotions. Furthermore, this formalism eliminates the tedious and error prone phase of translating algorithms into code.

Update dependencies is a language that supports high level thinking. However, there are a few disadvantages. Because update dependencies form an interpreted language, applications written in this language will run slower than those written in a compiled language. While update dependencies allow an explicit expression of the algorithms, it is necessary to enumerate implicit

cases, such as tests for key attribute instantiation. Thus to ensure consistency, the update dependencies must be completely defined. However, the need to express such complex dependency operations merits the use of such a language.

IMPLEMENTATION STRATEGY. Our implementation strategy has been planned to allow us to test the specification of the functional relationship between the MRP II and the CAD system early in the project.

Our first step has therefore been to define a formal language for specifying this functional relationship. The language allows us to specify the operations in and between the MRP II and the CAD system as an AI production system. We are currently evaluating this language.

The second step has been to implement an interpreter for the specification language. A first version of the interpreter has been implemented in Prolog. We are currently testing both the interpreter and the specification of the functional relationship between the MRP II and the CAD system under the control of one instance of the interpreter.

The third step is to integrate a remote procedure call facility into the interpreter. This will allow us to run functional copies of the MRP II and the CAD system under separate instances of the interpreter on the same machine.

The fourth step is to move the two interpreters to different machines by generalizing the remote procedure call facility to allow calls over the net.

An important aspect of this implementation strategy is that it allows early testing of the specification of the functional relationship between the MRP II and the CAD system. Furthermore, step three and four should not imply any changes in this specification, i.e. the distribution of information in the system should be transparent to the user.

Whereas this implementation strategy does allow us to test the specification of the functional dependencies within each system and between the two systems, it does not provide an integration of two actual systems.

If we were to actually build a "bridge-box" for two given systems, we would proceed as follows. First, we would identify the set of operations available to the users in each system. These operations should continue to be available to the users and the user interface should to the extend possible be kept unchanged. Second, we would identify the software procedures that support these operations. Third, through calls to the interpreter, we would insert a set of update dependencies between the operations available to the users and the software procedures supporting them. These update dependencies would capture calls of operations made by the users and would issue calls of the software procedures supporting them, i.e. the calls of the software components would be implied operations in the update dependencies. This approach would allow us to enforce consistency both within and between the two systems, by reusing the software components already available.

It is important to realize that the above approach keeps the user interfaces stable, reuses the software components that are already there, avoids redesign of existing databases, and solves the problem. However, it is also important to realize, that a certain amount of additional programming cannot be avoided.

CONCLUSIONS. The need for manufacturing systems integration has resulted in a CIM crusade in which several industrial and academic researchers are involved. This work suggests a staged approach, starting with MRP II as the nucleus of the system and CAD as the first "satellite." The similarity of functions dealing with the product definition and administraion and the large degree of data commonality between BOM of MRP II and CAD call for an attempt to streamline the operations in both systems.

The generation and maintenance of part master records and product structures initiated in the product engineering/design division of every typical manufacturing organization has been modelled and transformed to a set of logical rules. These rules have been translated into update dependencies, using a form of a rule-based expert system. Having completed the programming phase, the system is being tested for inconsistencies. Approval of the logical rules will be sought from industrial experts to ensure the applicability of the model in a real working environment. The functional design of the model has been extended to cover the transition of single level product structures to MRP II.

Future steps include the extension of the model over two databases on the same computer and later on two computers using remote operation calls.

It is clearly desirable to integrate existing pieces of software rather than developing new systems from scratch. However, we do not in this project intend to provide a "bridge-box" that will allow users to hook up any two MRP II and CAD systems. What we do provide is the knowledge needed to specify a "bridge-box" for two given systems, and an interpreter that can be used in implementing it.

ACKNOWLEDGEMENTS. The National Science Foundation (Grant No. DMC85-04922) and the Systems Research Center of the University of Maryland (Grant NSFD CDR-85-00108) are acknowledged for their funding of this research.

REFERENCES.

1. "The State of CIM", Daniels Appleton, Datamation, Vol. 30, pp 66-72. (1984)

2. "The Engineering Research Centers as a Tool for Change in the Culture and Attitudes of Academic Engineering", David C. Hazen, Presentation at the Second Meeting of the Steering Group on Systems Aspects of Cross Disciplinary Engineering, August 8, 1985.

3. "MRP II Providing a Natural 'Hub' for Computer Integrated Manufacturing Systems", Kenneth A. Fox, Industrial Engineering, Vol. 16, No. 10, pp. 44-50 (October 1984).

4. "The CIMS Database: Goals, Problems, Case Studies, and Proposed Approaches Outlined", Michael Melkanoff, Industrial Engineering, Vol. 16, No. 11, pp. 78-92 (November 1984).

Appendix A. Selected Update Dependency Operations

```
*
* Routine to Insert Part Records into CAD
*


sert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum))
>    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
       /\ nonvar(Buom) /\ nonvar(Spnum)
       /\ nonvar(Sbnum)
       /\ cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum).

>    var(Pnum),
       write('Part Number?'),
       read(Pnum),
       insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)).

>    var(Des),
       write('Description?'),
       read(Des),
       insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)).

>    var(Buom),
       write('Unit of Measure?'),
       read(Buom),
       insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)).

>    var(Dnum),
       insert(cadpart(Pnum,unknown,Dsize,Des,Buom,Spnum,Sbnum)).

>    var(Dsize),
       insert(cadpart(Pnum,Dnum,unknown,Des,Buom,Spnum,Sbnum)).

>    var(Spnum),
       insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,unknown,Sbnum)).

>    var(Sbnum),
       insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,unknown)).

>    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
       /\ nonvar(Buom) /\ nonvar(Spnum) /\ nonvar(Sbnum)
       /\ cadpart(Pnum,_,_,_,_,_,_)
       /\ ~(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)),
       write('Part Number Already Exists').

>    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
       /\ nonvar(Buom) /\ nonvar(Spnum) /\ nonvar(Sbnum)
       /\ ~(cadpart(Pnum,_,_,_,_,_,_))
       /\ ~(mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_)),
       add(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)),
       insert(cadrevs(Pnum,Rev,unknown,unknown,w,Dfname)),
       write('Part Has Been Added').

>    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
       /\ ~(cadpart(Pnum,_,_,_,_,_,_))
       /\ mrppmr(Pnum,Dnum,Dsize,Des,Buom,_,_,_,_,_,Spnum,Sbnum),
```

```
   add(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)).


Routine to Insert Revision Records into CAD


:rt(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname))

   nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
     /\ nonvar(Cstat) /\ nonvar(Dfname)
     /\ cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname).

   var(Pnum),
     write('Part Number?'),
     read(Pnum),
     insert(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

   var(Rev),
     write('New Revision Level?'),
     read(Rev),
     insert(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

   var(Dfname),
     write('Drawing File Name?'),
     read(Dfname),
     insert(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

   var(Estart),
     insert(cadrevs(Pnum,Rev,unknown,Eend,Cstat,Dfname)).

   var(Eend),
     insert(cadrevs(Pnum,Rev,Estart,unknown,Cstat,Dfname)).

   nonvar(Pnum)
     /\ ~(cadpart(Pnum,_,_,_,_,_,_)),
     write('Part Does Not Exist').

   nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
     /\ nonvar(Dfname)
     /\ cadrevs(Pnum,Rev,_,_,_,_)
     /\ ~(cadrevs(Pnum,Rev,Estart,Eend,_,Dfname)),
     write('Revision Level Already Exists').

   nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
     /\ nonvar(Dfname)
     /\ cadpart(Pnum,_,_,_,_,_,_)
     /\ ~(cadrevs(Pnum,Rev,_,_,_,_))
     /\ ~(mrprevs(Pnum,Rev,_,_,_)),
     add(cadrevs(Pnum,Rev,Estart,Eend,w,Dfname)),
     write('Revision Has Been Added').

   nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
     /\ nonvar(Dfname)
     /\ cadpart(Pnum,_,_,_,_,_,_)
     /\ ~(cadrevs(Pnum,Rev,_,_,_,_))
     /\ mrprevs(Pnum,Rev,Estart,Eend,Mstat),
     add(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).
```

```
**
** Routine to Release "working" CAD Revisions
**


releasework(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname))

->    var(Pnum),
        write('Part Number?'),
        read(Pnum),
        releasework(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

->    var(Rev),
        write('Revision Level?'),
        read(Rev),
        releasework(cadrevs(Pnum,Rev,Estart,Eend,Cstat,Dfname)).

->    nonvar(Pnum)
        /\ ~(cadpart(Pnum,_,_,_,_,_,_)),
        write('Part Number Does Not Exist').

->    nonvar(Pnum) /\ nonvar(Rev)
        /\ ~(cadrevs(Pnum,Rev,_,_,_,_)),
        write('Revision Level Does Not Exist').

->    nonvar(Pnum) /\ nonvar(Rev)
        /\ cadrevs(Pnum,Rev,_,_,_,_)
        /\ ~(cadrevs(Pnum,Rev,_,_,w,_)),
        write('Part Does Not Have Working Status').

->    nonvar(Pnum) /\ nonvar(Rev)
        /\ cadrevs(Pnum,Rev,Estart,Eend,w,Dfname)
        /\ cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)
        /\ ~(mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_)),
        insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,unknown,unknown,
        unknown,unknown,unknown,Spnum,Sbnum)),
        insert(mrprevs(Pnum,Rev,unknown,unknown,h)),
        remove(cadrevs(Pnum,Rev,Estart,Eend,w,Dfname)),
        add(cadrevs(Pnum,Rev,Estart,Eend,r,Dfname)),
        write('Revision Has Been Released').

->    nonvar(Pnum) /\ nonvar(Rev)
        /\ cadrevs(Pnum,Rev,Estart,Eend,w,Dfname)
        /\ mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_),
        insert(mrprevs(Pnum,Rev,Estart,Eend,h)),
        remove(cadrevs(Pnum,Rev,Estart,Eend,w,Dfname)),
        add(cadrevs(Pnum,Rev,Estart,Eend,r,Dfname)),
        write('Revision Has Been Released').



**
** Routine to Insert Part Master Records into MRP II
**

nsert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
```

```
    Lt,Spnum,Sbnum))

  nonvar(Pnum) ∧ nonvar(Dnum) ∧ nonvar(Dsize) ∧ nonvar(Des)
    ∧ nonvar(Buom) ∧ nonvar(Spnum) ∧ nonvar(Sbnum)
    ∧ mrppmr(Pnum,Dnum,Dsize,Des,Buom,_,_,_,_,_,Spnum,Sbnum).

  var(Pnum),
    write('Part Number?'),
    read(Pnum),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Des),
    write('Description?'),
    read(Des),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Buom),
    write('BOM Unit of Measure?'),
    read(Buom),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Puom),
    write('Purchasing Unit of Measure?'),
    read(Puom),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Cfuom),
    write('Unit of Measure Conversion Factor?'),
    read(Cfuom),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Scode),
    write('Source Code?'),
    read(Scode),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Lt),
    write('Lead Time?'),
    read(Lt),
    insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Dnum),
    insert(mrppmr(Pnum,inapp,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Dsize),
    insert(mrppmr(Pnum,Dnum,inapp,Des,Buom,Puom,Cfuom,Scode,Cost,
    Lt,Spnum,Sbnum)).

  var(Cost),

ert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,unknown,
    Lt,Spnum,Sbnum)).
```

```
->    var(Spnum),
        insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
        Lt,unknown,Sbnum)).

->    var(Sbnum),
        insert(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
        Lt,Spnum,unknown)).

->    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
        /\ nonvar(Buom) /\ nonvar(Puom) /\ nonvar(Cfuom) /\ nonvar(Scode)
        /\ nonvar(Cost) /\ nonvar(Lt) /\ nonvar(Spnum) /\ nonvar(Sbnum)
        /\ mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_)
        /\ ~(mprpmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
        Lt,Spnum,Sbnum)),
        write('Part Number Already Exists').

->    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
        /\ nonvar(Buom) /\ nonvar(Puom) /\ nonvar(Cfuom) /\ nonvar(Scode)
        /\ nonvar(Cost) /\ nonvar(Lt) /\ nonvar(Spnum) /\ nonvar(Sbnum)
        /\ cadpart(Pnum,_,_,_,_,_,_)
        /\ ~(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)),
        write('Part Number Already Exists').

->    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
        /\ nonvar(Buom) /\ nonvar(Puom) /\ nonvar(Cfuom) /\ nonvar(Scode)
        /\ nonvar(Cost) /\ nonvar(Lt) /\ nonvar(Spnum) /\ nonvar(Sbnum)
        /\ ~(mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_))
        /\ ~(cadpart(Pnum,_,_,_,_,_,_)),
        add(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
        Lt,Spnum,Sbnum)),
        insert(cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum)),
        insert(mrprevs(Pnum,Rev,Estart,Eend,Mstat)),
        write('Part Has Been Added').


->    nonvar(Pnum) /\ nonvar(Dnum) /\ nonvar(Dsize) /\ nonvar(Des)
        /\ nonvar(Buom) /\ nonvar(Puom) /\ nonvar(Cfuom) /\ nonvar(Scode)
        /\ nonvar(Cost) /\ nonvar(Lt) /\ nonvar(Spnum) /\ nonvar(Sbnum)
        /\ ~(mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_))
        /\ cadpart(Pnum,Dnum,Dsize,Des,Buom,Spnum,Sbnum),
        add(mrppmr(Pnum,Dnum,Dsize,Des,Buom,Puom,Cfuom,Scode,Cost,
        Lt,Spnum,Sbnum)).



**
** Routine to Insert Part Master Revision Data into MRP II
**


nsert(mrprevs(Pnum,Rev,Estart,Eend,Mstat))

->    nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
        /\ nonvar(Mstat)
        /\ mrprevs(Pnum,Rev,Estart,Eend,Mstat).

->    var(Pnum),
        write('Part Number?'),
        read(Pnum),
```

```
  insert(mrprevs(Pnum,Rev,Estart,Eend,Mstat)).

var(Rev),
  write('Revision Level?'),
  read(Rev),
  insert(mrprevs(Pnum,Rev,Estart,Eend,Mstat)).

var(Estart),
  write('Effectivity Start Date?'),
  read(Estart),
  insert(mrprevs(Pnum,Rev,Estart,Eend,Mstat)).

var(Eend),
  write('Effectivity End Date?'),
  read(Eend),
  insert(mrprevs(Pnum,Rev,Estart,Eend,Mstat)).

var(Mstat),
  write('Status Code?'),
  read(Mstat),
  insert(mrprevs(Pnum,Rev,Estart,Eend,Mstat)).

nonvar(Pnum)
  /\ ~(mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_)),
  write('Part Number Does Not Exist').

nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
  /\ nonvar(Mstat)
  /\ mrprevs(Pnum,Rev,_,_,_)
  /\ ~(mrprevs(Pnum,Rev,Estart,Eend,Mstat)),
  write('Revision Level Already Exists').

nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
  /\ nonvar(Mstat)
  /\ mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_)
  /\ ~(mrprevs(Pnum,Rev,_,_,_))
  /\ ~(cadrevs(Pnum,Rev,_,_,_,_)),
  add(mrprevs(Pnum,Rev,Estart,Eend,Mstat)),
  insert(cadrevs(Pnum,Rev,Estart,Eend,r,inapp)),
  write('Revision Has Been Added').

nonvar(Pnum) /\ nonvar(Rev) /\ nonvar(Estart) /\ nonvar(Eend)
  /\ nonvar(Mstat)
  /\ mrppmr(Pnum,_,_,_,_,_,_,_,_,_,_,_,_)
  /\ ~(mrprevs(Pnum,Rev,_,_,_))
  /\ cadrevs(Pnum,Rev,_,_,_,_),
  add(mrprevs(Pnum,Rev,Estart,Eend,h)).
```