(Draft) Proceedings of the 3rd International Conference on Data and Knowledge Systems for Manufacturing and Engineering. 1992.

Implementation of Rule Based Specifications for CIM Database Applications

P.R. Muro-Medrano
Electrical Engineering and Computer Science Department
University of Zaragoza, Spain

G. Harhalakis, C.P. Lin, L. Mark
Systems Research Center
University of Maryland
College Park, MD 20742, USA

Abstract

This paper focuses on the development of a methodology within a software environment for automating the rule based implementation of specifications of integrated manufacturing information systems. The specifications are initially formulated in a natural language and subsequently represented in terms of a graphical representation by the system designer. A new graphical representation tool is based on Updated Petri Nets (UPN) which we have developed as a specialized version of Colored Petri Nets (CPN). The application presented here deals with the control and management of information flow between Computer Aided Design, Process Planning, Manufacturing Resource Planning and Shop Floor Control databses.

1 Introduction

In a modern factory, besides *parts* being produced, there is also a tremendous amount of *data* being processed. For an efficient operation, it is necessary not only to control the manufacturing processes of products but also to manage and control the information flow among all the computerized manufacturing application systems that exist in a modern factory. The emphasis of most of the previous and current research projects is on individual aspects of CIM, such as RPI [HSU 87] on developing a global database framework, TRW [SEPE 87] on synchronizing the interface between application systems and distributed databases, and U. of Illinois [LU 86] on developing a framework to perform common manufacturing tasks such as monitoring, diagnostics, control, simulation, and scheduling. Their approach aims at developing a generic CIM architecture, creating a global database framework, or interfacing shop floor activities. However, the future in automation of modern factories will be based on a distributed environment which needs not only a generic database framework but also a controller, usually a knowledge rule-based system, to control the relationships between activities within all the computerized manufacturing application systems. Our approach is to develop such a control mechanism, in the form of a rule based system, for managing the information flow among all

the existing and new manufacturing application systems, and to fill the gap between the high level production management and the low level factory automation [HARH 90] [HARH 91]. Similar approach has been used in [DILT 91] which, different from ours, emphasizes on the design of an integrated database framework and lacks of a formal modeling tool for validation and implementation.

This paper first briefly presents a design methodology for transforming user specifications (company policies and expert rules) into executable computer code to control the information flow in a distributed environment with multiple databases. This methodology reflects the procedure to build a knowledge base serving as the control mechanism. It includes knowledge acquisition, graphical modeling, systematic validation and automated implementation. It features an enhanced graphic modeling tool - Updated Petri Nets (UPN) - which is capable of modeling database updates and retrievals, under specific constraints and conditions, and uses a hierarchical modeling approach. The emphasis of this paper is placed, however, on the automatic translation of the structural representation (UPN) into a rule specification language, which facilitates the implementation stage and reduces the design cycle of frequently changing knowledge rule-based systems.

This paper is structured as follows. The second section presents the overall design methodology of our INformation System for Integrated Manufacturing (INSIM), its specifications and architecture [HARH 91]. The third section discusses the UPN and its features. The fourth section describes the Update Dependencies language used for the implementation of the knowledge rule-based system. The fifth section details the implementation strategy and the translation procedure. The sixth section provides an example of the automatic translation between the UPN and UDL based on the rule specification in the CAD/CAPP/MRP II/SFC integrated system. The last section presents our conclusions with recommendations for future work.

2 Knowledge Base Design Methodology

Our research, aiming at linking product and process design, manufacturing operations and production management, focuses on the control of information flow between each of the key manufacturing applications at the factory level, including Computer Aided Design (CAD), Computer Aided Process Planning (CAPP), Manufacturing Resource Planning (MRP II), and Shop Floor Control (SFC) systems. This linkage between manufacturing application systems involves both the static semantic knowledge of data commonalities and the dynamic control of functional relationships. The common data entities, which form the basis of the integrated system, include: *Parts, Bills of Material* in CAD, *Parts, Bills of Material, Work Centers, Routings* in CAPP, *Parts, Bills of Material, Routings, Work Centers, Manufacturing Orders* in MRP II, *Parts, Routings, Work Centers, Manufacturing Orders* in SFC. The functional relationships deal with the inter-relationships of functions within those applications.

The design and maintenance of a Knowledge Based System (KBS) to control the functional relationships and information flow within the integrated system is a major task for the design of Knowledge Based Systems. Our design methodology for it is illustrated in [HARH 91]. It starts from user defined rule specifications, reflecting a specific company policy, which is then modeled using a special set of Colored Petri Nets - UPN (Updated Petri Net) and a hierarchical modeling methodology. The next step is to convert the UPN model into a set of General Petri

Nets (GPN) for validation purposes, and feed the results back to the user to resolve conflicting company rules and errors introduced during the modeling phase. After the model has been validated, a parser translates the UPN model into a rule specification language. The end result is a software package that controls the data flow and accessibility between distributed databases. In short, the input is a set of company rules and the output is an AI production system for controlling operations, accessibility and updates of data within the manufacturing applications involved.

3 Structured Modeling of the Domain Knowledge

3.1 Evolution of Updated Petri Nets

Petri Nets have been applied to most systems in representing graphically not only sequential but also concurrent activities [PETE 81] [MURA 89]. Because of their mathematical representation, they can be formulated into state equations, algebraic equations, and other mathematical models. Therefore, Petri Nets can be analyzed mathematically for the verification of system models and are ideal for modeling dynamically and formally analyzing complex dynamic relationships of interacting systems. Although General Petri Nets initially adopted in this research can in principle handle the modeling of the domain knowledge, it has become necessary to define more complex semantics in order to handle the increasing complexity of the domain knowledge, due to the involvement of more applications and their entities. Hence we have developed the Updated Petri Nets (UPN), which is a specialized type of the Colored Petri Nets (CPN) [JENS 87], and a hierarchical modeling methodology with a systematic approach for the synthesis of separate nets. The use of UPN allows the model designer to work at different levels of abstraction. Once we have this net we can selectively focus the analysis effort on a particular level within the hierarchy of a large model.

An UPN is a directed graph with three types of nodes: places which represent facts or predicates, primitive transitions which represent actions, and compound transitions which represent metarules (subnets). Enabling and causal conditions and information flow specifications are represented by arcs connecting places and transitions.

Formally, an UPN is represented as: $UPN =< P, T, C, I^-, I^+, M_0, I_o, MT >$, where:

1. P, T, C, I^-, I^+, M_0 represent the classic Color Petri net definition. They identify the part of the information system that provide the conditions for the information control. Only this part of the UPN net is used in the validation process. These terms are defined as follows [JENS 87]:

   - $P = \{p_1, ..., p_n\}$ denotes the set of places (represented graphically as circles). $T = \{t_1, ..., t_m\}$ denotes the set of primitive transitions (represented graphically as black bars). $P \bigcap T = \emptyset$ and $P \bigcup T \neq \emptyset$.
   - C is the color function defined from $P \bigcup T$ into non-empty sets. It attaches to each place a set of possible token-data and to each transition a set of possible data occurrence.
   - I^- and I^+ are negative and positive incidence functions defined on $P \times T$, such that $I^-(p,t), I^+(p,t) \in [C(t)_{MS} \rightarrow C(p)_{MS}]_L \ \forall (p,t) \in P \times T$ where S_{MS} denotes the set of all finite multisets over the non-empty set S, $[C(t)_{MS} \rightarrow C(p)_{MS}]$ the multiset

4 The Update Dependency Language, Syntax and Semantics

We have adopted a fairly new concept in systems integration, known as database interoperability. It is being realized through the development of the Update Dependency Language (UDL) in the Department of Computer Science, at the University of Maryland [MARK 87]. Database interoperability can be described as the concatenation of the schemata of each of the databases of the application systems, along with a rule set constructed for each separate database, called update dependencies. These update dependencies control inter-database consistency through inter-database operation calls. We propose the use of UDL as a special rule specification language, to be used for the implementation of our Knowledge Based System. The syntax and semantics of the language are formally presented in the following subsections.

4.1 UDL Syntax

For each relation and view defined in a relational database, the database designer defines procedures for the three database modifications insertion, deletion, and update. In addition, a set of application procedures for each relation may be defined, or as is the case in this paper, automatically generated by the translation from UPN to UDL.

Procedures have the following form:

$$
\begin{aligned}
&OR(A_1 = V_1, ..., A_n = V_n[; A_1 = W_1, ..., A_n = W_n]) \\
&\quad \longrightarrow C_1, O_{1,1}, ..., O_{1,n_1}. \\
&\quad \longrightarrow ... \\
&\quad \longrightarrow C_m, O_{m,1}, ..., O_{m,n_m}.
\end{aligned}
$$

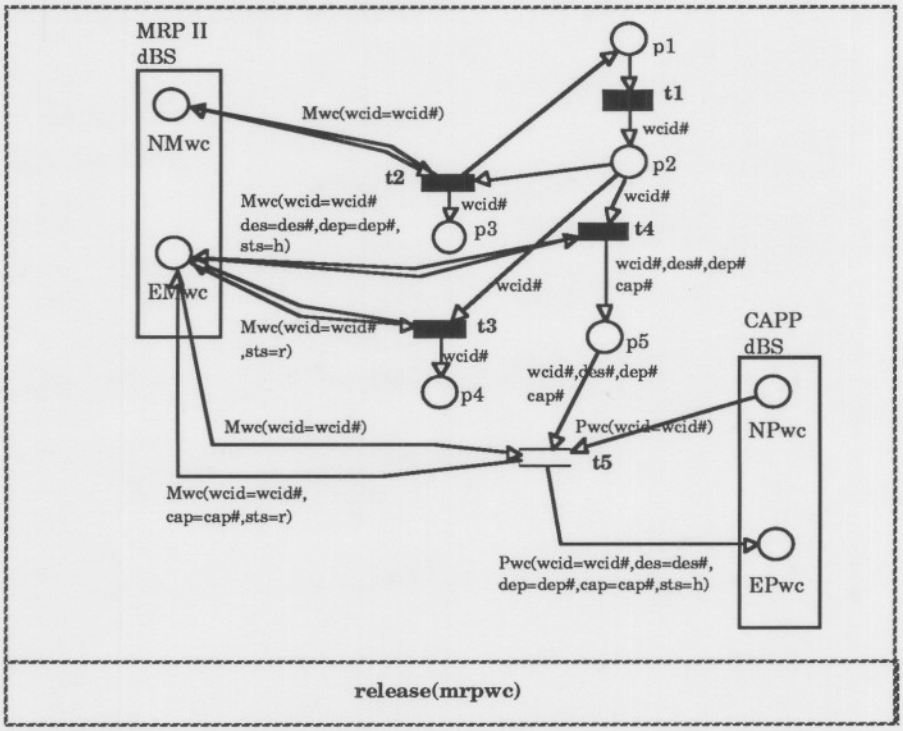
where [ ] indicates an optional element.

A procedure is uniquely identified by its operation type O and the name R of the base relation or view for which it is defined. The type of a modification procedure is either *insert*, *delete*, or *update*; the type of an application procedure is a *user-defined* name. The formal parameter list, required for all procedures, binds the values of relation R's attributes A_i to the variables V_i, $1 \leq i \leq n$. The *replacement* parameter list, used only in update procedures, binds the replacement values for relation R's attributes A_i to the variables W_i, $1 \leq i \leq n$.

As an example, an application procedure named `release`, is applied on the work center relation in the MRP II database and involves two modification procedures: `update` and `insert`. This example, which releases a work center record in MRP II, is shown in figure 2 and discussed in details below.

The *body* of a procedure consists of a set of procedure alternatives, each with the elements: a condition C_i, $1 \leq i \leq m$, on the database state; and a sequence of procedure invocations $O_{i,1}, ...O_{i,n_i}$, $1 \leq i \leq m$. *Conditions* are safe expressions formed through conjunction and negation of the following atoms (parenthesis are used to alter the default precedence of operators):

- *Tuple existence tests* with the form, $R(A_1 = V_1, ..., A_k = V_k)$, where R is the name of any base relation or view defined in the database, A_i, $1 \leq i \leq k$, are attribute names of R, and



t1: request and read wcid
t2: write error message and return
t3: write error message
t4: request other information
t5: update work center record in MRP II dBase with sts=r, and additional data
insert a work center record in CAPP dBase

p1: user starts the transaction
p2: wcid is provided
p3: work center ID does not exist in MRP II
p4: work center already has 'r' status in MRP II
p5: all the necessary data is provided
EMwc: existence of work center in MRP II DataBase
NMwc: non-existence of work center in MRP II DataBase
EPwc: existence of work center in CAPP DataBase
NPwc: non-existence of work center in CAPP DataBase

Figure 1: Subnet of the work center creation scenario "Release of a work center in MRP II".

extension of $[C(t) \rightarrow C(p)_{MS}]$ and $[\ldots]_L$ denote a set of linear functions (although, any linear function is allowed in the general color Petri net, only projections, identities and decolor functions have been needed so far in our models). The net has no isolated places or transitions.

- M_0 the initial marking is a funtion, such that: $M_0(p) \in C(p), \forall p \in P$.

2. I_o is an inhibitor function defined on $P \times T$, such that:
   $I_o(p,t) \in [C(t)_{MS} \rightarrow C(p)_{MS}]_L, \quad \forall (p,t) \in P \times T$.

3. $MT = \{mt_1, ..., mt_l\}$ denotes the set of compound transitions (represented graphically as blank bars), these are transitions which will be refined into more detailed subnets.

We have divided the representation of the domain knowledge in the following four groups: *Data, Facts, Rules, Metarules. Data* and relations between different data are used in relational database management systems. *Facts* are used to declare a piece of information about some data, or data relations in the system. The control of information flow is achieved by *Rules.* Here, we are considering domains where the user specifies information control policies using "if then" rules. Rules are expressed in UPN by means of transitions and arcs. Metaknowledge, in the form of metarules, is represented by net aggregation and hierarchical net decomposition (compound transition), and will be detailed below.

An example, which represents the release of a work center in MRP II, is explained in natural language below and is modeled in UPN, as shown in figure 1 to illustrate the corresponding component of UPN. Invoking the work center release transaction in MRP II triggers a set of consistency checks, which are as follows: the WC I.D. provided must exist in MRP II with hold status; all the required data fields should have been filled, and any data fields left out by users are requested at this stage. If all these checks are satisfied, the system changes the work center status code from 'hold' to 'released', and a skeletal work center record is automatically created in the work center file in CAPP, with its status set to 'working'.

Data : An example can be illustrated by the work center relation in MRP II with the record name as Mwc.

Facts : Facts in UPN will be represented by places and the tokens in the places. Facts can be seen in figure 1 where they are used to represent some user specifications (places $p_1, p_2, p_3, p_4, p_5, NMwc, EMwc, NPwc$ and $EPwc$).

Rules : Rules are expressed in UPN as the combination of two entities: transitions and the arcs with their associated functions connecting the transition with its input/output places. Arcs identify information flow and flow conditions.

Metarules : Metaknowledge and hierarchical net descriptions are represented by *Metarules* (expressed by compound transitions of the UPN) and mainly used in UPN as a mechanism to define subnets. They are used in two different directions to allow a structural and hierarchical composition of the domain knowledge: **Horizontal** metarules relate rules at the same level of abstraction and allow the aggregation of rules under specific criteria. For example, the relationship of rules shown in figure 1 is a horizontal metarule. **Vertical** metarules establish relationships between one rule and other rules which define knowledge at a lower level of abstraction and allow a structure of rules to form an abstraction hierarchy.

V_i , $1 \leq i \leq k$, are constants or variables. The relation, Mwc, used in the above example would have the following form: `Mwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap)`. Similarly, the tuple non-existence tests are reprepresented in the following form: $\sim R(A_1 = V_1, ..., A_k = V_k)$. A test of the non-existence of a work center record in MRP II is shown in the above example as: $\sim$ `Mwc(wcid=Wcid)`.

- *Comparisons* of the form, $X\ \theta\ Y$, where θ is comparison operator $(<, \leq, =, \geq, >)$ and X and Y are constants or variables. A comparison evaluates to true if the algebraic relation θ holds between X and Y. The *empty* condition. It always evaluates to true.

- *Negative or positive variable instantiation tests* with the form, var(V_i) or nonvar(V_i), where V_i, $1 \leq i \leq n$, are variables introduced in the head of the procedure. In the above example, `var(Wcid)` and `nonvar(Wcid)` are used to test the negative and positive instantiation of variable Wcid.

- *Existential quantification, exists* $V_1 ... V_n$ C. An existential qualification evaluates to true if there is at least one substitution of values V_i, $1 \leq i \leq n$ that satisfies the sub-condition C, which cannot contain any instantiation tests. There must be at least one occurrence of each V_i that is free in C.

Procedure invocations have one of the following forms:

- an *application procedure invocation* has the form (e_k and f_k are values of the respective attribute): $< user\ def.\ name > R(A_1 = e_1, ..., A_k = e_k[; A_1 = f_1, ..., A_k = f_k])$.

  E.g. `release Mwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap)`

- *insertion, deletion and update procedure invocations* have the forms:
  insert $R(A_1 = e_1, ..., A_k = e_k)$, *delete* $R(A_1 = e_1, ..., A_k = e_k)$, and
  update $R(A_1 = e_1, ..., A_k = e_k; A_1 = f_1, ..., A_k = f_k)$.
  E.g. `update Mwc(wcid=Wcid,sts=h;wcid=Wcid,cap=Cap,sts=r)`

- *physical insertion, deletion, and update invocations* have the forms:
  ins $R(A_1 = e_1, ..., A_n = e_n)$, *del* $R(A_1 = e_1, ..., A_n = e_n)$, and
  upd $R(A_1 = e_1, ..., A_n = e_n; A_1 = f_1, ..., A_n = f_n)$.

- primitive i/o operations for *read* and *write*, and the operation *fail* are also included in the update dependency formalism.

The procedure abstraction/encapsulation hierarchy enforced by the syntax for the update dependency formalism has three levels. The bottom level corresponds to the physical operations; the middle level corresponds to the modification procedures; and the top level corresponds to the application procedures.

4.2 UDL Semantics

The execution of a procedure can be depicted by an AND/OR graph. The *AND nodes* are those whose executions are tied together by an arc; the *OR nodes* are those whose executions are not tied together by an arc. Each execution of an OR node represents the execution of *one procedure alternative*. The ordered sequence (left-to-right) of executions of an AND node represents the

execution of the *elements of one procedure alternative*; the first represents the evaluation of the condition, and the following represent the executions of the invoked procedures. A *ROOT node* represents the execution of a user-invoked procedure. A *LEAF node* represents the evaluation of a condition, the execution of a physical insertion, deletion or update, or the execution of an i/o operation. An OR node *succeeds* if one of its executions succeeds. An AND node *succeeds* if the evaluation of its condition returns the value *TRUE* and the execution of each of the procedures it invokes succeeds.

When a procedure is invoked, then its *formal parameters are bound to the actual parameters.* The scope of a variable is one procedure. Conditions are submitted to the database system as queries, thus the order of evaluation of atoms is determined at run-time. The evaluation of a condition returns the value *TRUE* if the query corresponding to the condition returns a non-empty result; existentially quantified variables are bound to values that satisfy the query. The execution of a physical insertion, deletion or update, and the execution of an i/o operation always succeed. The selection of execution of procedure alternatives is non-deterministic and execution of procedure alternatives may be done in parallel. However, the effects of only one of the alternative will be seen when the procedure succeeds. Furthermore, while an alternative is executing it will only see database updates that have happened on its execution path; it will not see database updates from other alternatives that might be executing in parallel. If a procedure execution fails, i.e. none of its alternatives succeed, then the database is left completely unchanged by the procedure invocation. Conditions are submitted to the database system as queries, as mentioned above.

5 Translation of UPN to UDL

In this section, we would like to focus on the implementation of the user specifications. Once we have a structured and formal view of these specifications, we need to translate them to an execution language. Starting with an UPN we attempt to create a program capable of satisfying all specifications represented in that net.

User specifications do not necessarily need to be concerned with some problems which are already managed by the existing computer software technology. For example, database management systems are capable to deal with problems related to the concurrent access to the database; furthermore, if one update operation can not be successfully completed no part of that operation is performed. Therefore, these issues need not be part of the model

This section describes first the translation of particular features of UPN to UDL, and then proposes the translation procedure.

5.1 Data in UPN as UDL Relations

The information flowing through an UPN net can be atomic data, although this atomic information can be aggregated into more complex data structures. Atomic data and its data set can be translated to UDL as domains. For example, the data set of a work center status in MRP II (which can have only two different values h for hold, and r for released: $STS = \{r, h\}$) can be represented in UDL by a domain of character type. In UDL data structures are defined by a relation name and a tuple of data, which correspond to specific attributes specified in UPN: $R(A_1 = V_1, ..., A_k = V_k)$. An example of a work center record in MRP II in the form of a

UDL relation is shown below. It represents a work center `lt101` (wcid) which is a `lathe` (des), located in the `machining` (dep) department, having `h`(*hold*) status (sts), `na`(*not available*) state (ste), `null`(*unknown*) capacity (cap), `M12` resource code (res), and `null`(*unknown*) effectivity start date (esd).

```
Mwc(wcid=lt101,des=lathe,dep=machining,cap=null,sts=h,ste=na,res=M12,esd=null)
```

5.2 Facts in UPN as UDL Conditions

In order to verify whether a rule is enabled or not, it is necessary to verify if the precondition part of the rule matches with the status information in the system. Status information is represented by UPN places and their marking. Access to that information is specified in UPN by means of arcs and arc expressions. Two different types of status information can be distinguished:

Database status : Requires the access to a database record and the values of its attributes. This can be implemented by using the UDL relational form where the record is identified by the record id number. For example: In figure 1, the database check of work center *lt101* having a *hold* status, corresponds in UPN with an arc from the place $EMwc$ of the MRP database having the function $wcid = lt101, sts = h$. This is translated into UDL in the same form: `Mwc(wcid=lt101,sts=h)`. On the other hand the non-existence of the work center *lt101* corresponds in UPN with an arc from the place `NMwc` of the MRP II database having the function $wcid = lt101$, which can be translated into the UDL form of: $\sim$ `Mwc(wcid=lt101)`.

Reasoning process status : Generally corresponds to the intermediate states of an UDL application procedure. For example, places p_1 to p_5 in figure 1.

5.3 Database related arc conditions

The next step in the translation process is to identify UPN elements, which correspond to arc conditions directly relating to database places, in order to translate them into UDL elements. They will be translated into UDL checking conditions or modification procedures to access or modify the database. These elements are identified as follows:

- **Checking** a record. In UPN form, the database check is represented by a pair of input and output arcs, which have the same arc expression, linked between a transition and a database place. The check is implemented, as mentioned before, for database access. The case of a database place representing the non-existence of the record will be implemented using the UDL negative form. For example, transition t_3 in figure 1 has two arcs to and from place $EMwc$ (in the MRP II database) with the same arc expression: $wcid = wcid\#, sts = r$. This can be translated into UDL form as:
  ```
  Mwc(wcid=Wcid,sts=r)
  ```

- **Inserting** a record occurs when there is an arc from a database place to a transition which represents non-existence of a record, and another arc from the transition to a database place representing the existence of the same record. It is implemented using the UDL modification procedure `insert`(< *relation name* >(< *tuple spec* >)). For example, transition t_5 in figure 1 has one arc from place $NPwc$ and one to place $EPwc$ (in the MRP II database) with the arc expression $Pwc(wcid = wcid\#, des = des\#, dep =$

$dep\#, cap = cap\#, sts = w$). This can be translated into UDL form as:
`insert Pwc(wcid=Wcid, des=Des, dep=Dep, cap=Cap, sts=w)`

- **Deleting** a record from the database can be recognized when an arc stems from a database place representing the existence of a record to a transition, and another arc stems from the transition to a database place representing the non-existence of the same record. It is implemented using the UDL modification procedure as: `delete(<` *relation name* `>(<` *tuple spec* `>))`

- **Updating** a record in the database can be recognized when an arc stems from a database place representing the existence of a record, to a transition and another arc, in the reverse direction, but with different functions. It is implemented using the UDL modification procedure `update(<` *relation name* `>(<` *old tuple spec* `>;[<` *new tuple spec* `> ]))`. For example, transition t_5 in figure 1 implies an update to the record Mwc (in place $EMwc$) in the MRP II database that can be translated into UDL form as:
`update(Mwc(wcid=Wcid;wcid=Wcid,cap=Cap,sts=r))`

5.4 Requesting/printing information

The following is to identify UPN elements, which correspond to arc conditions directly relating to information input/output, to translate them into UDL i/o primitives operations for requesting or printing information to the user:

- **Requesting information** from the users. They can be recognized when a transition is a source transition, where some information that is leaving the transition through the outcoming arcs did not enter through any incoming arcs. This new information must be requested from the user. It is implemented using the UDL primitive operation `read (<` *domain variable* `>)`. For example, transition t_1 if figure 1 does not receive information from place p_1 but one needs to provide a work center identification number in variable $wcid\#$. This information must be provided by the user and can be implemented by: `read (Wcid)`.

- **Printing** a message to the users. They can be recognized when sink places, where some information arrives at the place through the incoming arcs, but does not leave the place through any outgoing arcs (generally because it has no outgoing arcs), appear in the nets. This information must be shown to the user. It is implemented using the UDL primitive `write('<` *place label text* `>',<` *domain variable* `>)`. If there is no domain variable, the label identifying the place is shown as `write('<` *place label text* `>')`. The last option may be used to show single error messages. For example, place p_4 in figure 1 can be translated as an error message for the work center identification provided in variable $wcid\#$: `write('Output in P4 for data: ' wcid#)`

5.5 Rules and Metarules as UDL Procedures

The following step corresponds to the translation of the transition set itself. UDL procedures provide a very powerful mechanism to represent if-then rules (transitions). So, UDL procedures will be used to represent subnets at any level of abstraction. The translation strategy follows what we call an **information driven approach** for the translation. The purpose of the

integrated manufacturing information system is to collect some information which is used to affect the external world or the system itself, this means that in each context (scenario) transitions can be differentiated by the information they need and the information they provide. Parameters of the procedure reflect the information that may be needed in a context. Using UDL negative **var** or positive **nonvar** variable instantiation tests the availability of this information in the context can be checked. Successive transition executions are made by recursive calls to the same procedure. We need to identify when the recursive call sequence finishes, this happens when the reasoning process has found some solution and the subnet evolution finishes. We can identify this by finding out when the outgoing arcs of a transition do not inject information in internal places (places related with decision process information). The actual parameters sent in each recursive call correspond to the information that is transmitted to the postconditions of the transition being executed.

5.6 Translation Procedure

The translation of UPN to UDL can be seen as another special "implementation" of Petri nets, specific for this application domain. This implementation of UPN is simpler than the implementation of a generic colored Petri net due to the constraints imposed by UPN over the general Petri net formalism (the variety of preconditions are highly constrained, rules are supposed to be well structured in metarules, manufacturing database domain related specifications, etc.). The purpose of the translation procedure is to generate efficient code in UDL, the language in which the specifications will be executed. To start the translation procedure, the UPN model must be provided. The procedure for translating one subnet into a piece of UDL code is detailed as follows:

1. Generate a UDL procedure heading, based on the UPN metarule name ($< O >$) and its corresponding database relation ($< R >$). The set of attribute names to be included in the procedure's formal parameter list is defined by the set of all attribute names that appear in the arc expressions of the subnet $(A_1, ..., A_m)$. The procedure head is:
   $< O >< R > (A_1 = V_1, ..., A_m = V_m)$
   Where $(V_1, ..., V_m)$ is the set of formal variables for which the values of attributes, $A_1, ..., A_m$, from relation $< R >$ are bound (these variable names can be the same as those in the UPN model).

   One UDL procedure is composed by several alternatives, one for each transition in the metarule subnet. The following steps must be done for each transition.

2. Conditions for the alternatives (preconditions of the transition) are defined by incoming arcs to the transition:

   (a) Recognize **checking** UDL elements, as explained in section 5.3. The conjunction of these checkings is a precondition for the procedure alternative:
   $< R > (A_m = V_m, ..., A_n = V_n)$

   (b) Find positive variable instantiations by looking at the variables in the arc expressions from the incoming arcs which do not belong to the database checkings recognized above $(Var_i, ..., Var_j)$, and generate a positive variable instantiation test for each one. The conjunction of these tests is another precondition:
   `nonvar(`V_i`)` $\wedge$ ... $\wedge$ `nonvar(`V_j`)`

(c) The rest of the formal variables have negative instantiation. Only the variables representing attributes that will provide information to the output places and are not coming from the input places $(V_x, ..., V_y)$ must be checked. Generate a negative variable instantiation test for each of them. The conjunction of these tests is another precondition: `var(`V_x`)` $\wedge$ ... $\wedge$ `var(`V_y`)`

3. Operations for the alternatives (postconditions of the transition) are defined by outgoing arcs from the transition. Each one of the following steps can produce new operations:

   (a) Recognize **input** and **output** UDL elements, as explained in section 5.4. For each input variable generate the appropriate input sequence: `write('Enter` $<$ *Text* $V_p >$`'), read(`V_p`),`

   (b) Recognize **deletion**, **insertion** and **update** UDL modification procedures, as explained in section 5.3 and generate the appropriate invocations:
   `delete(`$<$ *relation name* $>$`(`$<$ *tuple spec* $>$`))`
   `insert(`$<$ *relation name* $>$`(`$<$ *tuple spec* $>$`))`
   `update(`$<$ *relation name* $>$`(`$<$ *old tuple spec* $>$`;[`$<$ *new tuple spec* $>$`]))`

   (c) Generate a recursive call if any of the transition's output places, which is not a database place, is an input place to any transition within the subnet. Only the variables $(V_i, ..., V_j)$ which are used in the outgoing arc expressions connecting to the mentioned output places are used in the parameter list of the procedure call.
   $< O >< R > (A_i = V_i, ..., A_j = V_j)$

6 Example of Translating a single-procedure UPN subnet into one UDL procedure

In order to clarify the translation procedure, we return to the example shown in figure 1, which was used to illustrate the creation of UPN models in section 3.1. This net is *simple* because it does not require further refinement to create additional subnets. The goal now is to translate the UPN representation to the respective UDL code.

The name of the UPN is 'release Mwc' and the corresponding database records - work center record in MRP II and CAPP - are:
`Mwc (wcid,des,dep,cap,sts,ste,res,esd)` and `Pwc (wcid,des,dep,cap,sts)`

1. Procedure heading generation: $< O >\leftarrow$ `release` (metarule name) and $< R >\leftarrow$ `Mwc` (corresponding database record). Attribute names that appear in the arc expressions are: $wcid, des, dep, cap, sts$ and their corresponding variables $(wcid\#, des\#, dep\#, cap\#)$ are modified into the follow UDL variable syntax: `Wcid, Des, Dep, Cap`. The procedure heading becomes $\Rightarrow$

```
release Mwc (wcid=Wcid, des=Des, dep=Dep, cap=Cap)
```

2. Conditions for the alternatives:

   t_1 There is no connection with database places. This means there is no checking of the database. On the other hand, no positive variable instantiations are needed. The rest of the variables (`Wcid, Des, Dep` and `Sts`) have negative instantiations; however, there is only one outgoing arc connected with place p_2 with arc expression

```
release Mwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap)
⟶ var(Wcid),
    write('Enter wcid'),
    read(Wcid),
    release Mwc(wcid=Wcid).
⟶ nonvar(Wcid) ∧ ~Mwc(wcid=Wcid),
    write('Work center ID does not exist in MRP II, enter again', Wcid),
    release Mwc().
⟶ nonvar(Wcid) ∧ Mwc(wcid=Wcid,sts=r),
    write('Work center already has ''r'' status in MRP II', Wcid),
⟶ nonvar(Wcid) ∧ var(Cap) ∧ Mwc(wcid=Wcid,des=Des,dep=Dep,sts=h),
    write('Enter capacity'),
    read(Cap),
    release Mwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap).
⟶ nonvar(Wcid) ∧ nonvar(Des) ∧ nonvar(Dep) ∧ nonvar(Cap),
    update Mwc(wcid=Wcid,sts=h;wcid=Wcid,cap=Cap,sts=r),
    insert Pwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap,sts=w).
```

Figure 2: UDL code for the "Release of a work center in MRP II".

wcid#. This means that a work center identification number (variable `Wcid`) was not provided in the incoming arcs to the transition, but will be provided to the outgoing arc. The complete condition part is ⇒ `var(Wcid)`

t_2 It has incoming and outgoing arcs to $NMwc$ (MRP II database) with the same arc expression $Mwc(wcid = wcid\#)$. This is a checking for the non existence of `Mwc` with that specific work center identification number ⇒ `~Mwc(wcid=Wcid)`. It has another incoming arc with *wcid#* from p_2 providing the work center id. information which must be checked for positive instantiation ⇒ `nonvar(Wcid)`. The complete condition part is the conjunction of these conditions ⇒
`nonvar(Wcid) ∧ ~Mwc(wcid=Wcid)`

t_3, t_4, t_5 Similarly their complete condition are shown in figure 2.

3. Operations for the alternatives:

t_1 Variable's *wcid#* value needs to be requested (there is no incoming variables and variable *wcid#* is outgoing) ⇒
`write('Enter wcid'), read(Wcid),`. On the other hand, t_1 has an output place, p_2, which is an input place to transitions, t_2, t_3 and t_4. This means that the reasoning process is not completed yet and a recursive call is required. The parameters of this call are the ones required by the outgoing arcs ⇒ `release Mwc(wcid=Wcid)`

t_2 An output primitive can be easily recognized here: place p_3 is an output place (it is a sink place), the information in the arc expression, *wcid#*, and the text associated with the interpretation of p_3 must be displayed ⇒
`write('Work center ID does not exist in MRP II, enter again', Wcid),`
A recursive call is also required.

t_3, t_4, t_5 Similarly, operations for transitions t_3, t_4 and t_5 are recognized following the translation procedure and are shoen in figure 2.

7 Conclusions

The INformation Systems for Integrated Manufacturing (INSIM) design and maintenance methodology has been developed and implemented for generating knowledge based systems to effectively manage and control the information flow among CAD/CAPP/MRP II/SFC application systems. Its implementation strategy aims at facilitating the translation between UPN and UDL (as a rule specification language) and provides us a powerful tool to reduce the life cycle of developing knowledge bases. A prototype of the knowledge based system for integrating CAD/CAPP/MRP II/SFC application systems has been developed based on the proposed methodology with UPN and UDL technologies. This prototype has demonstrated the feasibility of our design methodology and has won considerable attention from both industry and other related research projects. The future work includes the incorporation of actual CAD, CAPP, MRP II, and SFC software packages and a database management system (ORACLE) as the next step of implementation.

References

[DILT 91] Dilts, D.M. and Wu, W. "Using Knowledge-Based Technology to Integrate CIM Databases ", *IEEE Transaction on Data and Knowledge Engineering, vol.3, no. 2, pp. 237-245* ,1991.

[HARH 90] Harhalakis, G., Lin, C., Hillion, H., and Moy, K., "Development of a Factory Level CIM Model", *Journal of Manufacturing Systems, vol. 9, no. 2, pp. 116-128*,1990.

[HARH 91] Harhalakis, G., Lin, C.P., Mark, L., and Muro, P., "Formal Representation, Verification and Implementation of Rule Based INformation Systems for Integrated Manufacturing (INSIM)", *Technical Report TR 91-19, Systems Research Center*, University of Maryland, College Park, 1991.

[HSU 87] Hsu, C., Angulo, C., Perry, A., and Rattner, L., "A Design Method for Manufacturing Information Management", *Proceedings of Conference on Data and Knowledge Systems for Manufacturing and Engineering, Hartford, Connecticut, pp. 93-102*,1987.

[JENS 87] Jensen, K., "Colored Petri Nets", *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986, Part I. Proceedings of an Advanced Course, Bad Honnef, 8-19. September 1986, pp. 248-299*, Edited by G. Goos and J. Hartmanis. Springer-Verlag Berlin Heidelberg 1987.

[LU 86] Lu, S.C.Y., "Knowledge-Based Expert System: A New Horizon of Manufacturing Automation", *Proceedings of Knowledge-Based Expert Systems for Manufacturing in the Winter Annual Meeting of ASME, Anaheim, California, pp. 11-23*,1986.

[MARK 87] Mark, L. and Roussopoulos, N., "Operational Specification of Update Dependencies", *Systems Research Center Technical Reprot No. SRC TR-87-37, University of Maryland*, 1987.

[MURA 89] Murata, T., "Petri Nets: Properties, Analysis and Applications", *Proceedings of The IEEE, vol. 77, no. 4, pp. 541-580*, April 1989.

[PETE 81] Peterson, J.L., "Petri Net Theory and the Modeling of Systems", Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[SEPE 87] Sepehri, M., "Integrated Data Base for Computer Integrated Manufacturing", *IEEE Circuits and Devices Magazine, pp. 48-54*, March 1987.