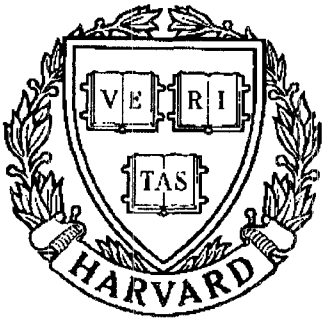
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Implementation of Rule Based Information Systems for Integrated Manufacturing

by G. Harhalakis, C.P. Lin, L. Mark, and P.R. Muro-Medrano

TR 92-12

Implementation of Rule Based Information Systems for Integrated Manufacturing

G. Harhalakis, C.P. Lin, L. Mark
Systems Research Center
University of Maryland

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

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 rule based implementation approach utilize the similarity of features between UPN and the general rule specification language used in the implementation. The automation of the translation of UPN to the rule specification language reduces considerably the life cycle for design and implementation of the system. 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. This provides an integrated information framework for Computer Integrated Manufacturing (CIM) systems.
Index Terms – Rule base, information system, computer integrated manufacturing, system modeling, knowledge verification, Petri nets, rule specification language, reasoning, language translation.

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 placed on individual aspects of CIM, such as RPI [1] on developing a global database framework, TRW [2] on synchronizing the interface between application systems and distributed databases, and U. of Illinois [3] 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 [4] [5]. A similar approach has been used in [6] which, different from ours, emphasizes on the design of an integrated database framework and lacks of a formal modeling tool for system validation and implementation.

As an example, an integrated manufacturing system with inter-related activities, which have precedence constraints between each other, was developed based on this methodology. The management and control of information flow is what differentiates our work from others, whose primary objective is to develop a consistent database framework or a standard communication protocol for data transformation. Our control mechanism, accompanied by existing distributed database management systems as shown in figure 1, can achieve a fully integrated manufacturing information system.

This paper 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 rule-based systems.

A rule specification language is needed for the implementation of the system. There exist a variety of programming languages and software development tools : LISP, PROLOG, PASCAL, and C for general purposel programming purposes; OPS5 for performing simulation, KEE for knowledge engineering, LOTOS (Language for Temporal Ordering Specification) by the ISO, for specifying data communication protocols, services and CIM system architectures, [7], and SAM by the National Institute of Standards and Technology (NIST) in its Automated Manufacturing Research Facility (AMRF) project for modeling data and activities in a manufacturing environment [8]. More recent research has focused on object-oriented programming and database management systems, which facilitate the development of new applications and improve system performance. ROSE developed by the Rensselaer Polytechnic Institute [9] [10] and KRON (Knowledge Representation Oriented Nets) by the University of Zaragoza [11] are some examples. The Update Dependencies Language (UDL) [12] was selected for our implementation due to the similarity of features between it and the UPN, and due to the advantage that UDL is designed specially for rule specifications and data updates. It consists of a rule set constructed for each separate database with its update and retrieval dependencies, to control inter-database consistency through inter-database operation calls.

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, which is more extensively described in [5]. The third section describes the Update Dependencies language used for the implementation of the rule-based system. The fourth section details the implementation strategy, the translation procedure, and provides examples of the automatic translation between UPN model and UDL code, based on an example of a rule specification in the CAD/CAPP/MRP II/SFC integrated system. The last section summarizes 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.

Our design methodology is depicted in figure 2 [5]. It starts from user defined rule specifications - '0', reflecting a specific company policy for the management and control of information flow, which is then modeled using a special set of Colored Petri Nets - UPN (Updated Petri Nets) and a hierarchical modeling methodology - '1'. The next step is to convert the UPN model into a set of General Petri Nets (GPN) - '3' - for model validation purposes, and to feed the results back to the user to resolve (i) conflicting company rules and (ii) errors introduced during the modeling phase - '4'. Once all subnet, each representing a set of user specification, have been validated, they will then been synthesized into a coherent net representing the integrated system specification - '2'. After the integrated model has been validated, a parser translates the UPN model into a rule specification language - '5'. 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.

2.1 Knowledge Acquisition (Company Policy)

The design of the model is based on the information flow established between all the manufacturing applications, namely CAD/CAPP/MRP II and SFC. The expert rules embedded in the knowledge base are extracted from company expertise, which can be obtained through a number of individual interviews and group meetings with experts from all manufacturing application systems to be integrated, and managers responsible for making company policy. Therefore, substantial effort may be required for gathering all expert rules to form the knowledge based system. However, since we are here to develop and demonstrate our design methodology, our prototype only includes limited rules extracted from our own industrial experience and other industries involved with this and other projects in the CIM Laboratory.

2.2 Structured Modeling of the Domain Knowledge

2.2.1 Evolution of Updated Petri Nets

Petri nets have been applied to most systems in representing graphically not only sequential but also concurrent activities [13] [14]. 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 Ggeneral 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 it, 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 Colored Petri Nets (CPN) [15], 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 and validation 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 provides 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 [15]:

   - $P = \{p_1, ..., p_n\}$ 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 $I^-(p,t), I^+(p,t) \in [C(t)_{MS} \rightarrow C(p)_{MS}]_L$ for $\forall (p,t) \in P \times T$, 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 extension of $[C(t) \rightarrow C(p)_{MS}]$ and $[...]_L$ a set of linear functions (although, any linear function is allowed in the general color Petri net, only projections, identities and decoloring functions have been used so far in our models).
   - The net has no isolated places or transitions:
     $\forall c \in P, \quad \exists t \in T : I^-(p,t) \neq 0 \vee I^+(p,t) \neq 0$ and
     $\forall t \in T, \quad \exists p \in P : I^-(p,t) \neq 0 \vee I^+(p,t) \neq 0$
   - M_0 the initial marking, is a funtion defined on P, such that:
     $M_0(p) \in C(p), \forall p \in P$.

Attibute	*Color set*	*DB data type*	*Description*
wcid	$WCID$	identification	identification number
des	DES	text	description
dep	DEP	text	department
cap	CAP	integer	capacity
sts	$MSTS$	$\{h, r\}$ (hold, release)	work center status code
ste	$MSTE$	$\{na, av\}$ (not avail., avail.)	work center state code
res	RES	text	resource code
esd	ESD	date	effectivity start date
Complete data structure for work centers in the MRP II database			
$Mwc(wcid, des, dep, cap, sts, ste, res, esd)$			

Table 1: Data information.

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 are 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 3. 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 : In an information system environment, the user needs to refer to atomic data, and establish relations between different data by structuring information into composed data objects. UPN allows the specification of atomic and composed data objects. As an example, let us suppose that a work center record in MRP II can be in one of two different status: r (release), h (hold). An atomic data object is illustrated by

the status set: $sts = \{r, h\}$. Furthermore, composed data objects used in UPN are a subset of the Cartesian product $S_1 \times S_2 \times \ldots \times S_n$, where S_i is a set of atomic data. An example of a composed data object is illustrated by the work center relation in MRP II with the record name as Mwc. Due to the specialized domain of this representation schema and database update, a special syntax is used to identify database relations: $< R > (< A_1 >, ..., < A_n >)$, where $< R >$ is the database relation and $< A_i >$ is the ith attribute of that relation. An example of the work center relation in MRP II is listed in table 1.

Facts : Facts in UPN will be represented by places and these tokens in the places. The fact asserted by one place is determined by the place name and its content (the colors of tokens in it). We represent facts about a work center record in MRP II with two places: $EMwc$, to describe the records that have been already introduced in the MRP II database, and $NMwc$, which expresses the negation of this fact. The UPN syntax of a fact within the database is $< R > (< A_1 >=< Val_1 >, ..., < A_n >=< Val_n >)$, where $< R >$ is the database relation, $< A_i >$ is the ith attribute of that relation, and $< Val_i >$ is the value or a corresponding variable of the ith attribute. These facts can be seen in figure 3 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. UPN provide different types of arcs:

Enabling arcs are directed arcs which connect a place/action with a transition/rule and define a precondition for the transition/action. They indicate which data must mark each place in order to enable a transition, as well as which data must be removed from that place on firing. In order to be closer to the formal view of the net, let us focus for example on transition t_5 in figure 3. Firstable, the color sets for the involved places and transitions must be identified:

$C(EMwc) = MWC = WCID \times DES \times DEP \times CAP \times MSTS \times MSTE \times RES \times ESD$
$C(NPwc) = WCID$
$C(EPwc) = PWC = WCID \times DES \times DEP \times PSTS$
$C(p_5) = WDDCS = WCID \times DES \times DEP \times CAP$
$C(t_5) = MWCSDD = WCID \times DES \times DEP \times CAP \times MSTE \times RES \times ESD \times DES \times DEP$
Color sets $WCID, DES, DEP, CAP, MSTS, MSTE, RES, ESD, PSTS$ are as specified in table 1.

Functions in I^- and I^+ are defined in terms of lambda expresions having the form $f(c) = \lambda(V)exp(c)$, where $c \in C(t)$. For transition t_5, V and $c \in MWCSDD$ can be represented as follows:

$V = wcid\#, des0, dep0, cap\#, mste0, res0, esd0, des\#, dep\#$ and
$c = wcid, deso, depo, cap, msteo, reso, esdo, des, dep$

The enabling arcs for t_5 are:

- $I^-(t_5, p_5) : exp = \begin{bmatrix} wcid\# \\ des\# \\ dep\# \\ cap\# \end{bmatrix}$, $\lambda(V)exp \in [MWCSDD_{MS} \rightarrow WDDCS_{MS}]_L$ such that
  $\lambda(V)exp(c) = wcid, des, dep, cap$
- $I^-(t_5, EMwc) : exp = [\ wcid = wcid\#\ ]$, $\lambda(V)exp \in [MWCSDD_{MS} \rightarrow MWC_{MS}]_L$ such that
  $\lambda(V)exp(c) = wcid, despo, depo, -, -, msteo, reso, esdo$
- $I^-(t_5, NPwc) : exp = [\ \ wcid\#\ \ ]$, $\lambda(V)exp \in [MWCSDD_{MS} \rightarrow WCID_{MS}]_L$ such that
  $\lambda(V)exp(c) = wcid$

Causal arcs are directed arcs which connect a transition/action with a place/fact and define a post-condition for the transition/action. Causal arcs describe modifications to be performed to the state of the net when a transition/rule is fired, and more concretely, they indicate which colors must be added to a place on firing. For example, these are the causal arcs for transition t_5 :

- $I^+(t_5, EMwc) : exp = \begin{bmatrix} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ cap = cap\# \\ sts = r \end{bmatrix} \lambda(V)exp \in [MWCSDD_{MS} \rightarrow MWC_{MS}]_L$ such that
  $\lambda(V)exp(c) = wcid, deso, depo, cap, r, msteo, reso, esdo$
- $I^+(t_5, EPwc) : exp = \begin{bmatrix} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ sts = w \end{bmatrix} \lambda(V)exp \in [MWCSDD_{MS} \rightarrow PWC_{MS}]_L$ such that
  $\lambda(V)exp(c) = wcid, des, dep, cap, w$

Checking arcs indicate which data must mark each place in order to enable a transition but not remove data. It can be represented as an enabling and causal arc together. The arcs connecting $EMwc$ and t_4 is an example being shown in figure 3. Additional **predicates** can be attached to the transitions which represent additional conditions applied on the values of variables used in the surrounding arcs. For example: a predicate, $cap\# \leq 1000$, may be attached to transition t_4 to assure that the capacity entered by the user is within a valid range.

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:

I^-		$t_{2,1}$	$t_{2,2}$	$t_{2,3}$	$t_{2,4}$	$t_{2,5}$
		E	WCID	MWCS	MWCSC	MWCSDD
NMwc	WCID	0	[$wcid\#$]	0	0	0
EMwc	MWC	0	0	$\begin{bmatrix} wcid = wcid\# \\ sts = r \end{bmatrix}$	$\begin{bmatrix} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ sts = h \end{bmatrix}$	$\begin{bmatrix} wcid = wcid\# \end{bmatrix}$
NPwc	WCID	0	0	0	0	[$wcid\#$]
EPwc	PWC	0	0	0	0	0
$p_{2,1}$	E	abs	0	0	0	0
$p_{2,2}$	WCID	0	[$wcid\#$]	[$wcid\#$]	[$wcid\#$]	0
$p_{2,3}$	WCID	0	0	0	0	0
$p_{2,4}$	WCID	0	0	0	0	0
$p_{2,5}$	WDDC	0	0	0	0	$\begin{bmatrix} wcid\# \\ des\# \\ dep\# \\ cap\# \end{bmatrix}$

Table 2: Negative incidence functions.

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 3 is a horizontal metarule. The formal representation of that subnet is specified by its incidence functions shown in tables 2 and 3, where:

$E = \{\varepsilon\}$
$MWCSC = WCID \times DES \times DEP \times CAP \times MSTE \times RES \times ESD \times DES \times DEP \times CAP$
$WDDC = WCID \times DES \times DEP \times CAP$

Vertical metarules establish relationships between one rule and other rules which define knowledge at a lower level of abstraction and allow the structure of rules to form an abstraction hierarchy.

A UPN consists of all the features described above and is currently used to represent the rules of the domain knowledge for manufacturing integration. The behavior of it is based on the firing of the transitions both sequentially and concurrently. A transition, which has in all its input places tokens satisfying the corresponding arc expressions and the predicates of the transition, is *enabled* and is subjected to be *fired* (executed). The results of firing one transition are the removal of tokens corresponding to the **enabling arc** functions from its input places and the addition of tokens corresponding to the **causal arc** functions to its output places. However, the tokens corresponding to the **checking arc** remain unchanged.

An example of transition firing is shown in figure 3. Transition t_4 is *enabled* if there exists one token in place p_2 with the color of $wcid\#$ and one token in place $EMwc$ with the color $Mwc(wcid = wcid\#, sts = h)$. After the firing of this transition, one token with the color of $wcid\#$ will be removed from place p_2 and one token with the color of $wcid\#, des\#, dep\#, cap\#, sts\#, ste\#, res\#, esd\#$ will be added into p_5.

I^+		$t_{2,1}$ E	$t_{2,2}$ WCID	$t_{2,3}$ MWCS	$t_{2,5}$ MWCSC	$t_{2,4}$ MWCSDD
NMwc	WCID	0	$[wcid\#]$	0	0	0
EMwc	MWC	0	0	$\begin{bmatrix} wcid = wcid\# \\ sts = r \end{bmatrix}$	$\begin{bmatrix} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ sts = h \end{bmatrix}$	$\begin{bmatrix} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ cap = cap\# \\ sts = r \end{bmatrix}$
NPwc	WCID	0	0	0	0	0
EPwc	PWC	0	0	0	0	$\begin{bmatrix} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ cap = cap\# \\ sts = w \end{bmatrix}$
$p_{2,1}$	E	0	abs	0	0	0
$p_{2,2}$	WCID	$[wcid\#]$	0	0	0	0
$p_{2,3}$	WCID	0	$[wcid\#]$	0	0	0
$p_{2,4}$	WCID	0	0	$[wcid\#]$	0	0
$p_{2,5}$	WDDC	0	0	0	$\begin{bmatrix} wcid\# \\ des\# \\ dep\# \\ cap\# \end{bmatrix}$	0

Table 3: Positive incidence function.

2.2.2 Hierarchical Modeling Approach

Generally speaking, any "company policy" starts from the specification of general global rules which describe aggregate operations for a given entity within the system. These rules are then further refined into more detailed specifications on a step-by-step basis, until no aggregate operations are left. In an attempt to assimilate this concept, a hierarchical modeling method using UPN has been developed, which allows the system designer to start from abstract global nets and continue with successive refinements until the desired degree of detail has been reached. In addition, company policies are usually provided for one entity at the time. Hence, a technique is needed to synthesize all scenarios to form a coherent net representing the unified company-wide policy for all entities in the system.

Some work in hierarchical representations using Petri nets has been done for various applications [16], [17], [18], [19]. A hierarchical modeling methodology facilitates the modeling task, and it incorporates:

Top-down stepwise refinement technique for the modeling of each scenario from an abstract and aggregate level to a detailed level. This approach necessitates the development of new Petri net modeling entities which include two types of transitions as mentioned in the previous section; one to represent primitive rules, and the other to represent metarules which can be further refined into subnets. The connections are represented by calls from one compound transition of the net at the abstract level to the subnets at the more detailed level, and an example is shown in figure 4. The transition, where the call was made, is formed by a calling net which contains one input transition (`ti`), one waiting place (`pw`), and one output transition

(to). There are, for each subnet being called, an arc connecting the input transiton and that subnet, and a returning arc back to its output transition. The interface between the input transition and the subnet being called is a place, representing the initiation of the subnet. The interface between the output transition and that subnet is a place, representing the satisfaction of the subnet.

Synthesis technique for synthesizing separate nets, which represent different scenarios of the system, to form a coherent net. Our modeling approach is capable of incorporating the modeling of the databases of the manufacturing application systems involved, using UPN, by defining the database states as global variables. We interface the application procedures (company policy) through the default modification procedure (system dependent) and places representing database states, and we synthesize nets through them systematically. More details can be found in [5]

2.3 Knowledge verification

One of the major objective of creating a KBS using Petri nets is the ability of validating the KBS mathematically and systematically. Completeness (dead-end rules, unfirable rules), consistency (redundant rules, subsumed rules, under-constrained rules), and conflicts, are the major issues in knowledge/rule validation [20], [21]. The incidence matrices of Petri nets representing the rule base can be used to perform some of these validation checks and verify them with the aid of specific domain knowledge. Several other analysis techniques for Petri nets, including, reachability trees, behavioral nets, and net invariants, are also used [14] [22]. The net invariants, which represent mutually exclusive conditions within the "company policy", can reveal logical conflicts in the specification of the original rules and possibly errors introduced during the modeling process. The reachability tree can be used to detect any deadlocks or inconsistencies in the model. The behavioral net can be used to detect redundancies in the net and is a useful tool for reducing the complexity of the model. The programs for computerizing these analysis methods have been developed and applied extensively. Some reduction rules [23] have also been investigated for reducing the complexity of nets prior to the analysis phase [5].

However, these analysis techniques were initially developed for Generalized Petri Nets (GPN), and do not apply to Colored Petri Nets (CPN), since the latter are characterized by a great diversity of linear functions that are associated to their arcs. Therefore, analysis algorithms for GPN that use integer matrices are not applicable to CPN. To overcome this obstacle, we have taken the approach of *unfolding* UPN into GPN before they are analyzed [5].

2.4 Implementation

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 [12]. 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 specifications of UDL and its features are described in the following section 3.

3 The Update Dependency Language, Syntax and Semantics

The Update Dependency Language (UDL) is a means to specify and control the semantics of a database under update. A set of update dependency procedures give a declarative operational specification of an update of a relation in terms of a set of alternative sequences of implied updates of the relation, and possibly of other relations, and specifies the conditions under which the implied updates must succeed for the original one to succeed.

The syntax and semantics of the language are formally presented in the following subsections. In section 4, in addition to the translation algorithm from UPN to UDL, we provide a number of examples of how the scenario used throughout this paper is translated into the formalism presented here.

3.1 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: `insert` and `update`. The example of releasing a work center record in MRP II, is shown in figure 5 and discussed in detail 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 V_i , $1 \leq i \leq k$, are constants or variables. The relation, Mwc, used in the above example represents the work center record in MRP II database and it contains the following attributes: $wcid, des, dep, cap, sts, ste, res, esd$. A tuple existence test evaluates to true if there exists at least one tuple in relation (or view) R, such that, for every instantiated variable V_i, the value of attribute A_i is equal to the value of V_i. A test of the existence of a work center record in MRP II with work center identification number `Wcid`, would have the following form:

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

  Every uninstantiated variable V_j, in this example `Des`, `Dep`, and `Cap`, will be instantiated as a result of the evaluation. The instantiated variables act as selection values and the uninstantiated variables act as either join or return value variables. 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:

  ```
  ~ Mwc(wcid=Wcid).
  ```

- *Comparisons* of the form, $X\ \theta\ Y$, where θ is a 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. The negative instantiation test evaluates to true if the variable V_i is not supplied in the invocation of the current procedure. The positive instantiation test evaluates to true if the variable V_i is supplied in the invocation of the current procedure. In the above example, `var(Wcid)` and `nonvar(Wcid)` are used to test the negative and positive instantiation of the 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)

  $< user\ \ defined\ \ name > R(A_1 = e_1, ..., A_k = e_k[; A_1 = f_1, ..., A_k = f_k])$.

  In the above example, the application procedure involved is:

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

- *insertion and deletion procedure invocations* have the forms:

  insert $R(A_1 = e_1, ..., A_k = e_k)$ and *delete* $R(A_1 = e_1, ..., A_k = e_k)$, respectively.

  In the above example, the insertion procedure involved is:

  ```
  insert Pwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap,sts=w)
  ```

- *update procedure invocations* have the form:

  update $R(A_1 = e_1, ..., A_k = e_k; A_1 = f_1, ..., A_k = f_k)$.

  In the above example, the update procedure involved is:

  ```
  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.

  In the above example, the primitive i/o operations involved include:

```
write('Enter wcid')
read(Wcid)
```

The procedure abstraction/encapsulation hierarchy enforced by the syntax of the update dependency formalism is illustrated in Figure 6. There are three levels in the hierarchy. 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. Notice that physical insertion, deletion, and update invocations on a base relation **R** are only allowed from insertion, deletion, and update procedures on **R**, respectively.

Notice that physical insertion, deletion, and update, *ins*, *del*, and *upd*, respectively, on a relation **R** can only be invoked from within insertion, deletion, and update procedures on the relation **R**, respectively. Furthermore, physical insertion, deletion and update, are not available on views; procedures for views are specified through the invocation of insertion, deletion and update procedures on the base relations the views are defined from. Finally, procedures may call each other and may call themselves recursively.

In the algorithm and examples in section 4, we utilize the procedures at the application and modification procedure levels only; we assume that the DBMS has provided the implementation of modification procedures, which work as the corresponding physical operations. In other words, we have assume that procedures *insert,.delete*, and *update* are working as operations *ins,.del*, and *upd*, respectively.

3.2 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 executions 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 occurred 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.

4 Translation of UPN to UDL

In this section, we 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, then proposes the translation procedure, and finally demonstrates the generation of UDL codes with examples.

4.1 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: $STS = \{r, h\}$) is 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). (It is reminded that general work center record in MRP II is represented as `Mwc(wcid,des,dep,cap,sts,ste,res,esd)`) :

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

4.2 Facts in UPN as UDL Conditions

In order to verify whether a rule is enabled or not, it is necessary to verify that 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: information about the database status and information about the reasoning process status.

Database status : Requires access to a database record and reading the values of its attributes. This is implemented by using the UDL relational form where the record is identified by the record id number.

For example: In figure 3, the database check of work center *lt101* with a *hold* status, corresponds in UPN to an arc from the place $EMwc$ of the MRP II database, with 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 to the UPN place `NMwc` of the MRP II database, with the function $wcid = lt101$; this can be translated into the UDL form of: $\sim$ `Mwc(wcid=lt101)`

Reasoning process status : Generally corresponds to the states of an UDL application procedure. For example, places p_1 to p_5 in figure 3.

4.3 Database Related Arc Conditions in UPN as UDL Checkings and Modification Procedures

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 are 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 is implemented using the UDL negative form. For example, transition t_3 in figure 3 has two arcs to and from place $EMwc$ (in the MRP II database) with the same arc expression: $wcid = wcid\#, sts = r$. This is 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 3 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 is translated into UDL form:

  ```
  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: `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 a different function. It is implemented using the UDL modification procedure `update(`< *relation name* >`(`< *old tuple spec* >`;[`< *new tuple spec* >`]))`. For example, transition t_5 in figure 3 implies an update to the record Mwc (in place $EMwc$) in the MRP II database that is translated into UDL form as:

  ```
  update(Mwc(wcid=Wcid;wcid=Wcid,cap=Cap,sts=r))
  ```

4.4 Requesting/Printing Information in UPN as UDL Primitive i/o Operations

The next step 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. Thus requesting information from or printing information to the user can be achieved. The primitive operations are identified as follows:

- **Requesting information** from the user. This is detected when a transition is a source transition, where some information that is leaving the transition through the outgoing arc(s) did not enter through any incoming arc(s). 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 3 does not receive information from place p_1. Instead one needs to provide a work center identification number in the variable *wcid#*. This information must be provided by the user and is implemented by:
  ```
  read (Wcid)
  ```
  For better legibility, a message like the following can be printed to prompt the user:
  ```
  write ('Input the value for the variable wcid')
  ```

- **Printing** a message to the user. This is detected when sink places appear in the net. Some information arrives at such a place through the incoming arc(s), but does not leave the place through any outgoing arc(s), generally because it has no outgoing arcs. 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 3 is translated as an error message for the work center identification provided in variable *wcid#*:
  ```
  write('Output in P4 for data:  '  wcid#)
  ```
  or, if the place has an associated label:
  ```
  write('work center already exists:'  wcid#)
  ```

4.5 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). As a first approach, each transition of an UPN net could be easily implemented by a separate UDL procedure. This approach for the translation of transitions is general and simple but it presents several problems. An important problem is that some additional local variables are required to represent the completed firing of each transition within the transition set.

Secondly, the approach does not make use of some important programming capabilities available in UDL, such as the use of procedures (application rules), and recursion. This would result in an inefficient implementation.

For example, in order to implement transition t_3 from figure 3 as a composed operation, we need a new variable, varP2, to test if the value Wcid is in place $P2 \Longrightarrow$

```
Release-Transition-t3 Mwc(wcid=Wcid,sts=Sts)
⟶ (varP2 = Wcid) ∧ Mwc(wcid=Wcid,sts=r),
    write('Work center already has ''r'' status in MRP II').
```

On the other hand, UDL provides a way to implement a set of related rules in the form of a composed rule. Also, procedures and procedure calls are typical decomposition mechanisms used in UDL programs and recursion is also available. Finally, UPN allows for a horizontal aggregation of related transitions that belong to a subnet. As an alternative to the first approach, our strategy is to implement a *set* of related rules as *one* UDL procedure. UDL procedures are used to represent subnets at any level of abstraction. It is no longer necessary to use additional local variables (other than the formal parameters of the procedures) to implement the transition status of the net execution.

To take all of these features into account, we follow what we call an **information driven approach**. 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 require and the information they provide. Parameters of the procedure reflect the information that may be needed in a context. Using the UDL negative var or positive nonvar variable instantiation tests, the availability of this informationm can be checked.

However, only one transition is executed in each operation call. A situation in which the outgoing arcs of a transition go to places internal to the subnet, which means that the subnet will continue firing the other transitions, is implemented by making a recursive call to the UDL procedure (subnet). Therefore, successive transition executions are made by recursive calls to the same procedure. We need to identify when the recursive call sequence has finished. This happens when none of the other transitions in the subnet can be enabled by the output of the firing transition. We can identify this by determining when the outgoing arcs of a transition do not connect to internal places (places can enable the other transitions within the subnet).

Procedure parameters transfer data from one call (transition execution) to the following one. The actual parameters sent in each recursive call correspond to the information that is transmitted to the postconditions of the transition being executed.

We need now to discuss the problem of how to deal with conflicts in the net (for example when two or more transitions are simultaneously enabled). This problem is solved by making use of the UDL alternative procedure capability, presented in section 3.2.

Conflicts are resolved by identifying a successful path, through trying different evolution alternatives which correspond to different transition firing sequences. Although this may appear as an "ad hoc" solution, it is sufficient for our problem domain.

4.6 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 added constraints imposed by UPN over the general Petri net formalism. Examples of such added constraints include: the variety of preconditions that are highly constrained, rules that are supposed to be well structured in metarules, specifications that are related to a manufacturing database domain. The overall purpose of the translation procedure is to generate an 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:

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 formal parameter list of the procedure 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 the 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 taken for each transition.

1. Conditions for alternatives (preconditions of transitions) are defined by incoming arc(s) to a transition:

   (a) Recognize **checking** UDL elements, as explained in section 4.3. 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:
   $\texttt{nonvar}(V_i) \wedge \ldots \wedge \texttt{nonvar}(V_j)$

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

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

   (a) Recognize **input** and **output** UDL elements, as explained in section 4.4. For each variable, that needs to be provided from the user, generate the appropriate input sequence ($< Text\ V_p >$ that corresponds to the interpretation of the attribute name bound by V_p in the database record tables):
   `write('Enter <`*Text* V_p`>'), read(`V_p`),`
   For each output variable generate: `write('<`*place label text*`>')`

   (b) Recognize **deletion**, **insertion** and **update** UDL modification procedures, as explained in section 4.3 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) Write the calls for all UDL application procedures associated with the transition. The recognition of UDL application procedure calls is based on the discussion in section 2.2.2.

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

4.7 Generation of UDL Code

The implementation of the knowledge based system is based on the translation from the UPN subnets (which are designed, validated, and refined according to the system specifications collected) into UDL code. There are two types of UPN subnets to be translated: the first, a single-procedure subnet which involves only one application procedure; the second, a multi-procedure subnet which involves more than one application procedure with procedural calls among subnets. Each application procedure has to be translated into one UDL code, following the translation procedure discussed in section

4.6, including the application procedure calls in the second case. Examples of translations for both single-procedure UPN subnets and multi-procedure UPN subnets are detailed in the following sections.

4.7.1 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 3, which was used to illustrate the creation of UPN models described in section 2.2.1. 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 work center record in CAPP - are described below (a more detailed description of Mwc is given in table 1):

Work center record in MRP II: `Mwc (wcid,des,dep,cap,sts,ste,res,esd)`
Work center record in CAPP: `Pwc (wcid,des,dep,cap,sts)`

Translation procedure

1. Procedure heading generation:
   $< O >\leftarrow$ `release` (metarule name)
   $< R >\leftarrow$ `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 following UDL variable syntax: `Wcid, Des, Dep, Cap`.
   The procedure heading becomes $\Longrightarrow$
   ```
   release Mwc (wcid=Wcid, des=Des, dep=Dep, cap=Cap)
   ```

2. Conditions for the alternatives:

   t_1 There is no connection with database places (rows $NMwc$ and $EMwc$ in I^- and I^+ are 0). This means there is no checking of the database. The column of transition t_1 in I^- shows that there is only one incoming arc connected with place p_1 with no variables attached to the arc expression. This means that no positive variable instantiations are needed. The rest of the variables (`Wcid, Des, Dep` and `Sts`) have negative instantiations; however, the column of transition t_1 in I^+ shows that there is only one outgoing arc connected with place p_2 with arc expression $wcid\#$. This means that in the incoming arcs to the transition a work center identification number (variable `Wcid`) was not provided, but will be provided to the outgoing arc. In order to reduce the code, only this test is really needed: `var(Wcid)`. The complete condition part is $\Longrightarrow$
   ```
   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 $\Longrightarrow$ `~Mwc(wcid=Wcid)`. It has another incoming arc with $wcid\#$ from p_2 providing the work center id. information this must be checked for positive instantiation $\Longrightarrow$ `nonvar(Wcid)`. There is no more outgoing information for the arc because the arc expression to place p_1 has no variables. This means that no negative instantiation test is necessary. The complete condition part is the conjunction of these two conditions $\Longrightarrow$

```
nonvar(Wcid) ∧ ~Mwc(wcid=Wcid)
```

t_3 Similarly, the complete condition is $\Longrightarrow$

```
nonvar(Wcid) ∧ Mwc(wcid=Wcid,sts=r)
```

t_4 Similarly, the complete condition is $\Longrightarrow$

```
nonvar(Wcid) ∧ var(Cap) ∧ Mwc(wcid=Wcid,des=Des,dep=Dep,sts=h)
```

t_5 Similarly, the complete condition is $\Longrightarrow$

```
nonvar(Wcid) ∧ nonvar(Des) ∧ nonvar(Dep) ∧ nonvar(Cap) ∧
Mwc(wcid=Wcid,sts=h)
```

3. Operations for the alternatives:

t_1 Column t_1 from I^- and I^+ shows that variable $wcid\#$ needs to be requested (there are no incoming variables and variable $wcid\#$ is outgoing) $\Longrightarrow$

```
write('Enter wcid'),
read(Wcid),
```

No other UDL elements (output, deletion, creation or update) can be recognized. However, transition 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 (in this case only `Wcid`) $\Longrightarrow$

```
release Mwc(wcid=Wcid)
```

t_2 An output primitive can be easily recognized here: place p_3 is an output place (or a sink place), thus the information in the arc expression, $wcid\#$, and the text associated with the interpretation of p_3 must be displayed $\Longrightarrow$

```
write('Work center ID does not exist in MRP II, enter again', Wcid),
```

As before, a recursive call is required, in this case with no call parameters (arc expression outgoing to place p_1 has no variables) $\Longrightarrow$

```
release Mwc()
```

t_3 Only an output statement is needed to display the information in the arc expression, $wcid\#$, and the text associated to the interpretation of p_4 $\Longrightarrow$

```
write('Work center already has ''r'' status in MRP II', Wcid),
```

No new call is needed because the output place p_4 is not connected to any other transition.

t_4 The input for variable Cap is required and then a recursive call is made with the information for the wcid, des, dep, cap and sts parameters $\Longrightarrow$

```
write('Enter capacity'),
read(Cap),
release Mwc (wcid=Wcid,des=Des,dep=Dep,cap=Cap).
```

t_5 An update modification procedure can be identified because there is an arc coming from the database place $EMwc$ with a different function ($Mwc(wcid = wcid\#)$) to the one that is going back to $EMwc$ ($Mwc(wcid = wcid\#, des = des\#, dep = dep\#, cap = cap\#, sts = r)$) $\Longrightarrow$

```
update Mwc(wcid=Wcid,sts=h;wcid=Wcid,cap=Cap,sts=r),
```

It also has an associated procedure call $\Longrightarrow$

```
insert Pwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap,sts=w).
```

The final UDL code resulting from this translation is shown in figure 5.

4.7.2 Example of Translating a Multi-Procedure UPN Subnet into UDL Procedures

An UPN subnet, which has been designed using a top-down refinement technique into a set of subnets, each representing one UDL application procedure, has to be translated into more than one piece of UDL code segments. An example of this kind is the removal of a work center record from MRP II presented here. MRP II is the execution function in most companies and is the sole center for the procurement and allocation of resources, and in turn, is the function through which equipment is phased out or removed from the system. When the removal operation is invoked in MRP II, the following system checks are initiated. A check is made to see that the work center being removed exists in MRP II. The status of the work center is not relevant to the operation. In addition, all routings maintained by the MRP II routing module are checked. If any routing utilizing this work center exist and are on 'hold' or 'release' status in CAPP, the operation fails and a message to this effect is displayed. The reason is that work centers which are utilized by active routings, cannot be removed. If the above checks are satisfied, the work center is removed from the databases of MRP II, CAPP and SFC. The above specification is first modeled in UPN at the abstract level as shown in figure 7 and then further refined down to a more detailed level, as shown in figure 8. The complete net involves three subnets, which are translated to three UDL procedures: one major procedure (procedure no. 1) removes the work center via MRP II and two other procedures check the MRP II and the CAPP databases (see the dashed boxes in figure 8). The top-down refinement technique used was discussed in section 2.2.2.

The goal now is to translate the UPN representations to the respective UDL codes. Following the same translation procedure for all the subnets involved, three UDL application procedures are generated as shown below (the UDL code is shown in figure 9).

(1). UPN subnet no. 1

During the translation of operations $t2$, the following two application procedures called by it have to be satisfied, before any other modification procedures can be implemented $\Longrightarrow$

```
check.1 Mwc (wcid=Wcid), check.2 Prout (wcid=Wcid,psts=Psts),.
```

Three deletion modification procedures can be identified: an arc coming from the database place $EMwc$ with the expresion $Mwc(wcid = wcid\#)$ and another one going to the database place $NMwc$ with the same expression (same is the case for CAPP and SFC) $\Longrightarrow$

```
delete Mwc(wcid=Wcid), delete Pwc(wcid=Wcid), delete Swc(wcid=Wcid).
```

(2). UPN subnet no. 2

During the translation of operation $t_{2,4}$, we observe that the output is a place $p_{2,3}$ which represents the interface with the higher level subnet. This place will receive a token as long as all the pre-conditions are satisfied. Therefore, no operation is required here.

(3). UPN subnet no. 3

During the translation of operation $t_{2,7}$, there are two negative checkings, which are represented by the inhibitor arcs, for the non-existence of any routing `EProut (wcid=Wcid,psts=h)` using that specific work center identification number and bearing an `h` or `r` status $\Longrightarrow$

$\sim$ `EProut (wcid=Wcid,psts=h)` and $\sim$ `EProut (wcid=Wcid,psts=r)`.

Similarly, the complete condition is the conjunction of all related conditions $\Longrightarrow$

`nonvar(Wcid)` $\wedge$ $\sim$ `EProut (wcid=Wcid,psts=h)` $\wedge$ $\sim$ `EProut (wcid=Wcid,psts=r),`

5 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. This knowledge base design methodology is fairly generic in that it can be applied to generate knowledge based systems for other applications as well. 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. Depending on the rule specification language used, this design methodology can be applied in the same way with a modified Petri Nets translator. 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. Furthermore, we will also focus on the developement of new techniques for knowledge verification.

References

[1] C. Hsu, C. Angulo, A. Perry, and L. Rattner, "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.

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

[3] S.C.Y. Lu, "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.

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

[5] G. Harhalakis, C.P. Lin, L. Mark, and P. Muro, "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.

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

[7] F. Biemans, and P. Blonk, "On the Formal Specification and Verification of CIM Architectures Using LOTOS," *Computers in Industry, vol. 7, pp. 491-504*,1986.

[8] S.Y.W. Su, "Modeling Integrated Manufacturing Data With SAM*," *Computer, vol. 19, no. 1, pp. 34-49*, 1986.

[9] M. Hardwick, and D. Spooner, "The ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications," *IEEE Transactions on Knowledge and Data Engineering, pp. 285-289*, 1989.

[10] D. Spooner, M. Hardwick, and et. al., "The evolution of ROSE: An Engineering Object-Oriented Database System," *Pro. of Conf. on CIM, RPI, pp. 16-23*, 1990.

[11] P.R. Muro, J.L. Villarroel, J. Martinez, and M. Silva, "A Knowledge Representation Tool for Manufacturing Control Systems Design and Prototyping," *INCOM'89, 6th IFAC/IFIC/IFORS/IMACS Symposium on Information Control Problems in Manufacturing Technology*, Madrid, Spain, September 1989.

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

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

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

[15] K. Jensen, "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.

[16] R. Valette, "Analysis of Petri Nets by Stepwise Refinements," *Journal of Computer and System Sciences 18, pp 35-46*, 1979.

[17] I. Suzuki, and T. Murata, "A method for stepwise refinement and abstraction of Petri nets," *Journal of Computer System Science, vol. 27, pp. 51-76*, 1983.

[18] Y. Narahari, and N. Viswanadham, "A Petri Net Approach to the Modeling and Analysis of Flexible Manufacturing Systems," *Annala of Operations Research, vol. 3, pp. 381-391*, 1985.

[19] M. D. Jeng, and F. DiCesare, "A Review of Synthesis Techniques for Petri Nets," *Proceedings of IEEE Computer Integrated Manufacturing Systems Conference, RPI*, May 1990.

[20] T.A. Nguyen, W.A. Perkins, T.J. Laffey, and D. Pecora, "Knowledge Base Validation," *AI Magazine, summer, pp. 67-75*. 1987.

[21] B. Lopez, P. Meseguer, and E. Plaza, "Knowledge Based Systems Validation: A State of the Art," *AI Communications, Vol.3, No. 2, pp. 58-72*. June, 1990.

[22] J. Martinez, and M. Silva, "A Simple and Fast Algorithm to Obtain All Invariants of A Generalised Petri Net," *Second European Workshop on Application and Theory of Petri Nets, pp. 301-310*, 1982.

[23] K.H. Lee, and J. Favrel, "Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets," *IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 2*, 1985.

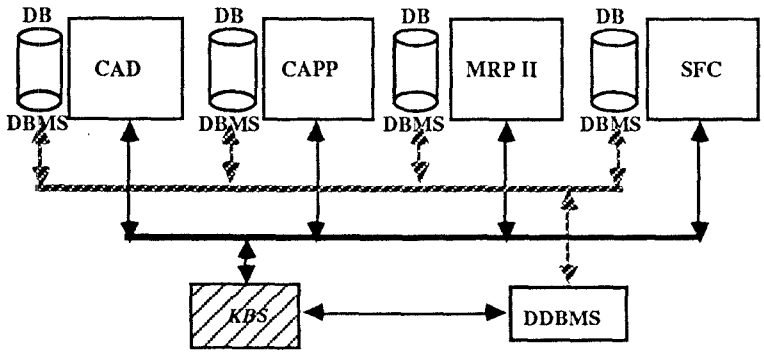


Figure 1: Information Flow Architecture for Manufacturing

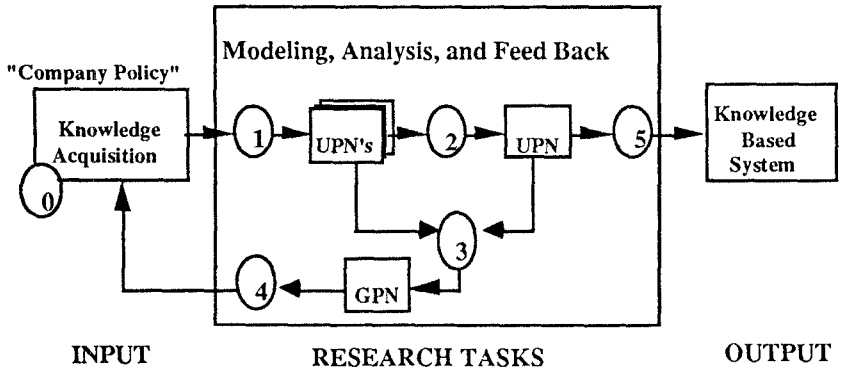
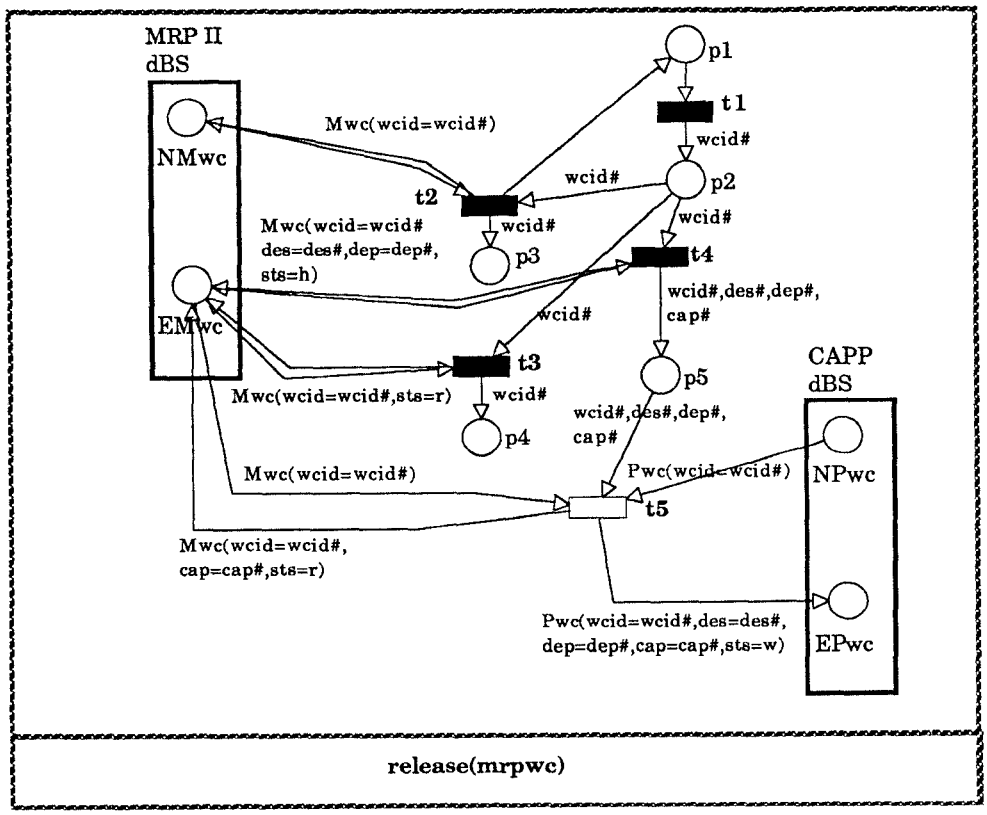


Figure 2: Knowledge Base Design Methodology



t1: request and read wcid
t2: write error message and restart
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 dBase
NMwc: non-existence of work center in MRP II dBase
EPwc: existence of work center in CAPP dBase
NPwc: non-existence of work center in CAPP dBase

Figure 3: Subnet of the work center creation scenario "Release of a work center in MRP II".

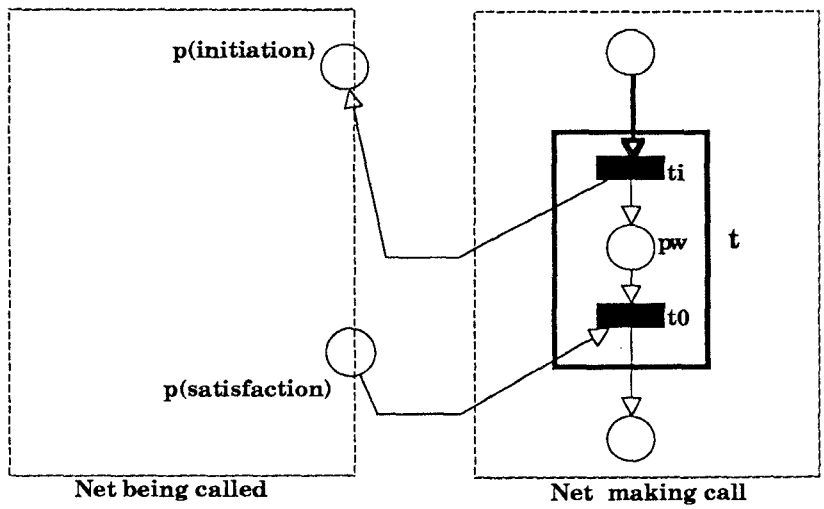


Figure 4: Example of call between UPN sub-nets

```
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 5: UDL code for the "Release of a work center in MRP II".

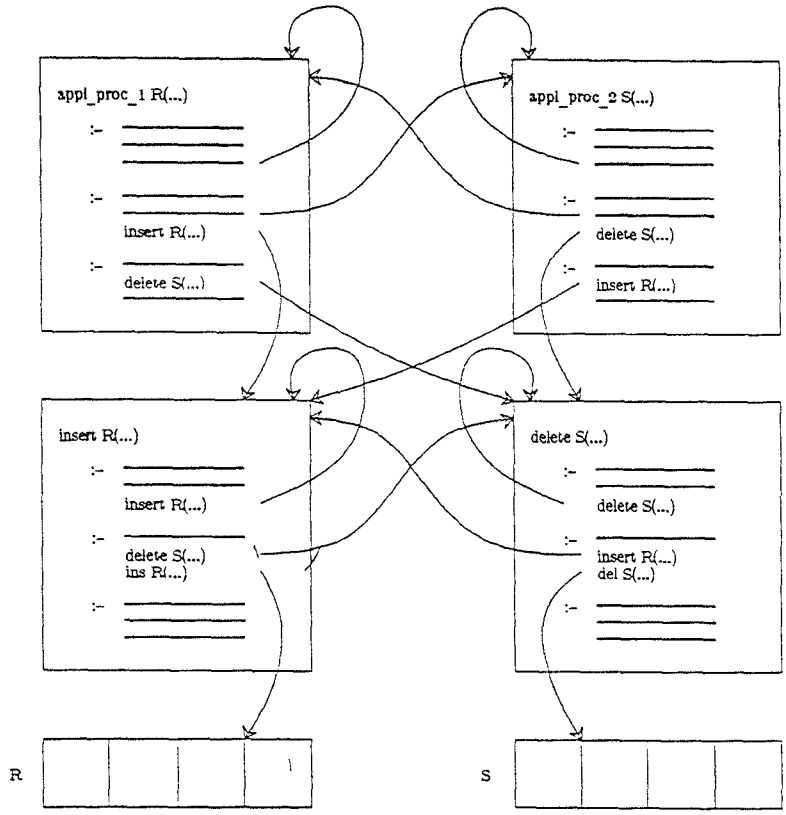


Figure 6: Procedure abstraction/encapsulation hierarchy.

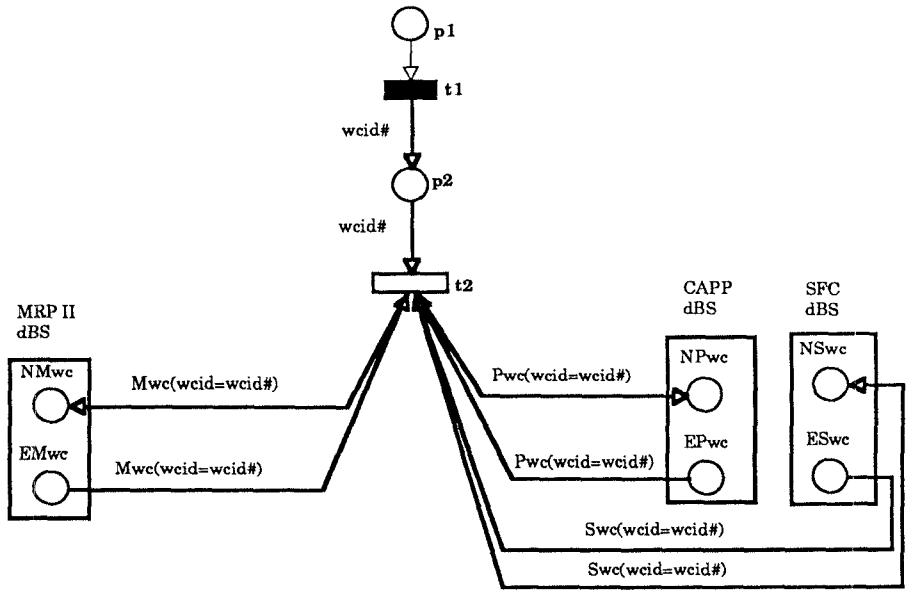


Figure 7: Subnet of the work center deletion scenario "Deletion of a work center in MRP II".

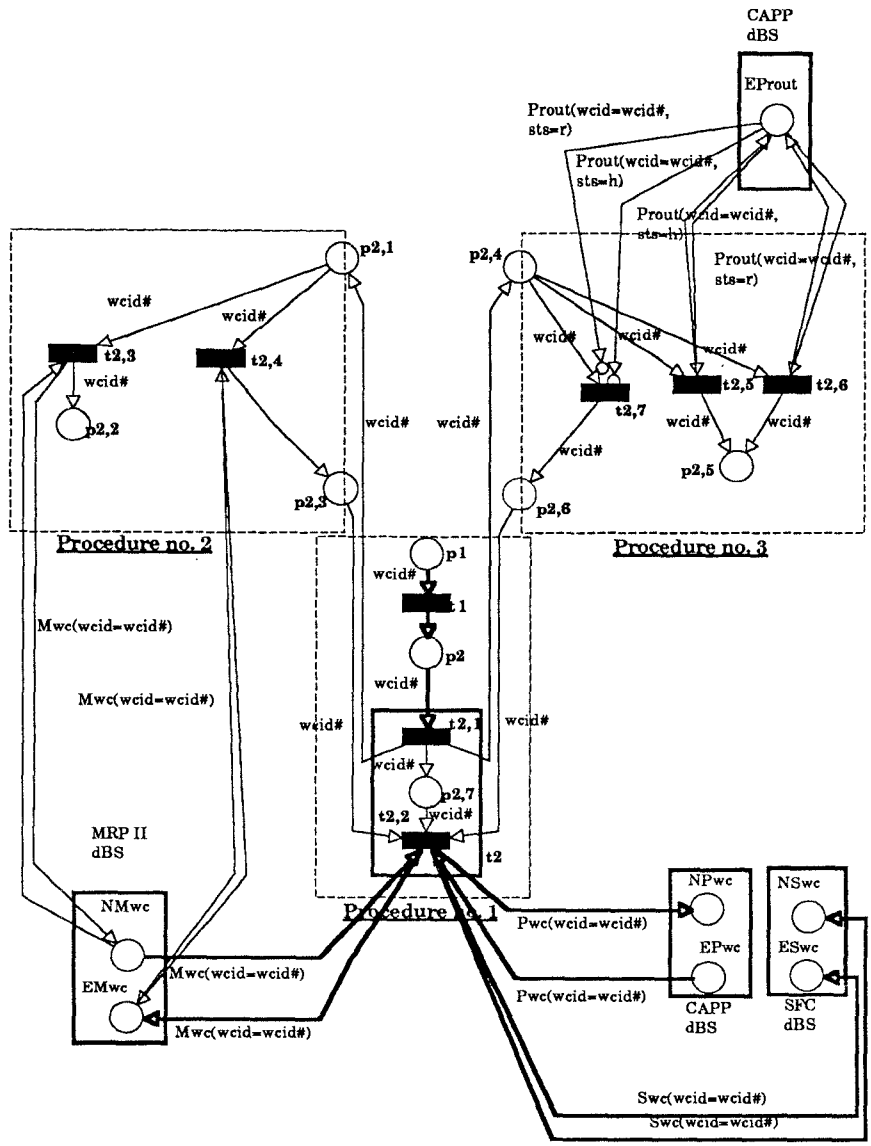


Figure 8: Subnet of the work center deletion scenario "Deletion of a work center in MRP II".

```
remove Mwc(wcid=Wcid)
⟶ var(Wcid),
    write('Enter wcid'),
    read(Wcid),
    remove Mwc(wcid=Wcid).

⟶ nonvar(Wcid),
    check1.rmv.mwc Mwc (wcid=Wcid),
    check2.rmv.mwc Prout (wcid=Wcid,psts=Psts),
    delete Mwc(wcid=Wcid),
    delete Pwc(wcid=Wcid),
    delete delete Swc(wcid=Wcid).

check1 Mwc(wcid=Wcid)
⟶ nonvar(Wcid) ∧ ~Mwc(wcid=Wcid),
    write('Work center ID does not exist in MRP II', Wcid).

⟶ nonvar(Wcid) ∧ Mwc(wcid=Wcid).

check2 Prout(wcid=Wcid)
⟶ nonvar(Wcid) ∧ nonvar(Wcid) ∧ EPwc (wcid=Wcid,psts=h),
    write('Work center is in use by active process plans', Wcid).

⟶ nonvar(Wcid) ∧ nonvar(Wcid) ∧ EPwc (wcid=Wcid,psts=r),
    write('Work center is in use by active process plans', Wcid).

⟶ nonvar(Wcid) ∧ ~ EProut (wcid=Wcid,psts=h) ∧ ~ EProut (wcid=Wcid,psts=r).
```

Figure 9: UDL code for the "Delete of a work center in MRP II".