Design, Verification and Implementation of Rule Based Information Systems for Integrated Manufacturing (INSIM)

by C-P. Lin
Advisor: G. Harhalakis
Design, Verification and Implementation
of Rule Based Information Systems
for Integrated Manufacturing
(INSIM)

by
Chang-Pin Lin

Dissertation submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1991

Advisory Committee:
  Dr. George Harhalakis, Chairman/Advisor
  Dr. D.K. Anand
  Dr. A.R. Hevner
  Dr. A.T. Jones
  Dr. P.A. Ligomenides
  Dr. L. Mark
  Dr. J.C.S. Yang
ABSTRACT

Title of Dissertation: Design, Verification and Implementation of Rule Based Information Systems for Integrated Manufacturing (INSIM)

Chang-Pin Lin, Doctor of Philosophy, 1991

Dissertation directed by: Dr. G. Harhalakis, Associate Professor
Department of Mechanical Engineering
University of Maryland, College Park

Full control and management of information flow in an automated factory has not yet been achieved, mainly because of the data inconsistencies and lack of established functional relationships among different manufacturing application systems. Research toward CIM has concentrated on the computerization of individual functions of manufacturing, such as computer aided design and shop floor control, and the integration of data relations, such as global database frameworks and distributed database management systems. A much needed mechanism to manage and control the information flow among all of the manufacturing application systems based on company-specific and company-wide policies and procedures, is proposed here. The goal is to achieve a fully integrated manufacturing management system. The INformation System for Integrated Manufacturing (INSIM) reflects a design methodology to build a knowledge base to serve as the information control mechanism. The methodology includes the collection of rules (knowledge acquisition), their graphical modeling, systematic model validation and automated implementation to an operating production system. This design methodology 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. Finally, a prototype knowledge based system written in Update Dependencies Language (UDL) - a special rule specification language - was implemented as a result of direct and automatic translation of UPN.
DEDICATION

To my parents and my wife
Acknowledgement

I wish to express my sincere gratitude to Professor Harhalakis, my advisor, whose guidance, wholehearted support, and advise throughout this research project is invaluable. I am also grateful to: Dr. Mark of the Computer Science Department for providing me with an Update Dependency Language interpreter and his valuable comments; Dr. Muro at the University of Zaragoza in Spain for his collaboration and dedication to this research project.

In addition, I wish to thank the other members of my dissertation committee. Drs. Anand (Mechanical Engineering Department), Hevner (business school), Jones (National Institute of Standards and Technology), Ligomenides (Electrical Engineering Department), and Yang (Mechanical Engineering Department), for their time, valuable comments, and suggestions.

I also like to thank all the members in the Computer Integrated Manufacturing Laboratory, Amy, Anshu, Ashu, Dave, George, Howard, Raj, Rakesh, Satish, Sudhanshu, Tom, with their cheer. Special thanks to Rakesh who have given me tons of help, support and suggestions.

Further, I wish to express my appreciation to the Systems Research Center and Department of Mechanical Engineering for providing financial support during this project.

Finally, I want to thank my family in Taiwan and my wife, Tase-Rong, for their patience, contribution and moral support
Contents

1 Introduction .................................. 1
   1.1 Background and Objectives .............. 1
   1.2 Research Issues .......................... 3
   1.3 Contributions of this work .............. 5
   1.4 Structure of the Dissertation .......... 6

2 Literature Survey ............................. 7
   2.1 CIM System Architectures and SFC .......... 7
   2.2 Database Frameworks for CIM ............. 9
   2.3 Knowledge Based Expert Systems for CIM . 11
   2.4 Systems Modeling Tools .................. 11
      2.4.1 IDEF and SADT ....................... 12
      2.4.2 GRAI ................................ 13
      2.4.3 Petri Nets ......................... 14
   2.5 Conclusion ................................ 15

3 Research Approach ............................ 16
   3.1 Introduction ............................. 16
3.1.1 CIM System Specification and Architecture .............. 17
3.1.2 Role of Each Application System .......................... 18
3.1.3 Multi-Database versus Central Database .................. 21
3.1.4 Overall CIM Information Flow Architecture ............... 22

3.2 System Design Methodology ................................. 23
3.2.1 Knowledge Acquisition (Company Policy) ................. 24
3.2.2 Modeling of the Domain Knowledge ....................... 27
3.2.3 Knowledge Verification ................................. 28

3.3 Implementation ............................................. 29

4 Updated Petri Nets (UPN) for Information Systems ........... 30
4.1 Introduction ............................................... 30
4.2 Evolution of Updated Petri Nets ............................ 31
4.3 Updated Petri Nets (UPN) Formalism ....................... 31
4.3.1 Data ................................................. 35
4.3.2 Facts ............................................... 36
4.3.3 Rule specification .................................... 39
4.3.4 Metarule specification ............................... 47
4.4 Modeling Approach ......................................... 48
4.4.1 Top-down Stepwise Refinement Technique ............... 48
4.4.2 Synthesis Technique .................................. 54
4.5 Conclusions ............................................... 59

5 Knowledge Verification ......................................... 60
5.1 Introduction ............................................. 60
5.2 Conversion of UPN to Generalized Petri Nets ................. 63
  5.2.1 Petri Net Formalism ................................ 64
  5.2.2 Abstraction ........................................ 65
  5.2.3 Unfolding ......................................... 66
5.3 General Net Analysis Methods and Properties ................. 69
5.4 Checking Completeness ................................... 71
5.5 Checking Consistency .................................... 72
5.6 Domain Knowledge Verification ............................ 75
5.7 Structural Verification of Complex Rules .................... 77
5.8 Example ................................................ 78
5.9 Conclusion .............................................. 82

6 Translation of UPN to Update Dependencies Language (UDL) 83
  6.1 Introduction ........................................... 83
  6.2 The Update Dependency Language, Syntax and Semantics .... 84
    6.2.1 UDL Syntax ...................................... 85
    6.2.2 UDL Semantics ................................... 90
  6.3 Feature Correspondence Between UPN and UDL ............... 92
    6.3.1 Data in UPN as UDL Relations .................... 92
    6.3.2 Facts in UPN as UDL Conditions .................. 93
    6.3.3 Database Related Arc Conditions in UPN as UDL Checking
         and Modification Procedures ....................... 93
6.3.4 Requesting/Printing Information in UPN as UDL Primitive i/o Operations ................................. 95
6.3.5 Rules and Metarules as UDL Procedures ......................... 96

6.4 Translation Procedure .............................................. 99
6.5 Generation of UDL Code ........................................... 101
  6.5.1 Example of Translating a Single-Procedure UPN Subnet into one UDL Procedure ...................... 102
  6.5.2 Example of Translating a Multi-Procedure UPN Subnet into UDL Procedures ......................... 106

6.6 Conclusions ......................................................... 111

7 Software Development and Application .......................... 112
  7.1 Software Development for the Design of KBS .................. 113
    7.1.1 Enhanced Petri Net Editor ................................. 113
    7.1.2 Petri Net Exploder and Synthesizer ...................... 113
    7.1.3 Petri Net Analyzer ........................................ 116
    7.1.4 Translator Between UPN to UDL ............................ 118
  7.2 Example .......................................................... 119
  7.3 Applications ..................................................... 126

8 Conclusions and Future work ...................................... 128

Bibliography .......................................................... 130

Appendices ............................................................ 139
  A Specification of CAD/CAPP/MRP II/SFC Integrated System . 139

vii
List of Tables

4.1 Data structures and color sets of work centers. 37
4.2 Positive incidence functions. 50
4.3 Negative incidence functions. 50
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Islands of automation</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>NIST CIM system architecture</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Data commonalities between CAD, CAPP, MRP II, and SFC</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>Definition of a generic shop floor control system</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Information flow architecture for manufacturing application systems</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>Knowledge base design methodology</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Subnet of the work center creation scenario: &quot;Release of a work center in MRP II&quot; with initial marking</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>UPN representation of the rule &quot;Release work center from MRP II&quot;</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>UPN graph of the scenario: &quot;Creation of a work center via MRP II&quot; at an abstract level with initial marking</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of a subnet and a calling protocol</td>
<td>53</td>
</tr>
<tr>
<td>4.5</td>
<td>Partially refined UPN of the scenario &quot;Creation of a work center via MRP II&quot; with initial marking</td>
<td>55</td>
</tr>
<tr>
<td>4.6</td>
<td>UPN graph of the scenario: &quot;Removal of a work center via MRP II at an abstract level with initial marking</td>
<td>57</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.7</td>
<td>UPN graph of the synthesized scenario: &quot;Creation and removal of a</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>work center via MRP II&quot; at an abstract level with initial marking</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>Procedural levels in Database Management Systems</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Staged model verification.</td>
<td>62</td>
</tr>
<tr>
<td>5.2</td>
<td>Conversion of a UPN rule to GPN.</td>
<td>67</td>
</tr>
<tr>
<td>5.3</td>
<td>Simple redundant rules.</td>
<td>73</td>
</tr>
<tr>
<td>5.4</td>
<td>Subsumed Rules.</td>
<td>74</td>
</tr>
<tr>
<td>5.5</td>
<td>Free choice rules.</td>
<td>75</td>
</tr>
<tr>
<td>5.6</td>
<td>Conflicting rules.</td>
<td>76</td>
</tr>
<tr>
<td>5.7</td>
<td>Complicated redundant rules.</td>
<td>78</td>
</tr>
<tr>
<td>5.8</td>
<td>Reduced net.</td>
<td>78</td>
</tr>
<tr>
<td>5.9</td>
<td>Unfolded UPN of the subnet: &quot;Release of a work center in MRP II&quot;.</td>
<td>79</td>
</tr>
<tr>
<td>5.10</td>
<td>Marked PN of the subnet: &quot;Release of a work center in MRP II&quot;.</td>
<td>81</td>
</tr>
<tr>
<td>5.11</td>
<td>Reachability tree generated for the net in figure 5.10.</td>
<td>81</td>
</tr>
<tr>
<td>6.1</td>
<td>Procedure abstraction/encapsulation hierarchy.</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>AND/OR graph</td>
<td>90</td>
</tr>
<tr>
<td>6.3</td>
<td>UDL code for the scenario &quot;Release of a work center in MRP II&quot;.</td>
<td>106</td>
</tr>
<tr>
<td>6.4</td>
<td>Subnet of the scenario &quot;Deletion of a work center in MRP II&quot;.</td>
<td>107</td>
</tr>
<tr>
<td>6.5</td>
<td>Subnet of the scenario &quot;Deletion of a work center in MRP II&quot;.</td>
<td>108</td>
</tr>
<tr>
<td>6.6</td>
<td>UDL code for the scenario &quot;Deletion of a work center in MRP II&quot;.</td>
<td>110</td>
</tr>
<tr>
<td>7.1</td>
<td>Computer screen of the Petri net editor.</td>
<td>114</td>
</tr>
<tr>
<td>7.2</td>
<td>Flow chart for the net explosion program.</td>
<td>115</td>
</tr>
</tbody>
</table>
7.3 Flow chart for the net synthesis program. 116
7.4 Flow chart for the abstraction and unfolding of UPN models program. 117
7.5 Flow chart for the consistency and conflict verification program. 118
7.6 Flow chart for the translation between UPN and UDL. 119
7.7 UPN of scenario: "Creation of a work center via MRP II". 122
7.8 UPN of scenario: "Creation of a work center in MRP II". 123
7.9 UPN of scenario: "Release of a work center in MRP II". 124
7.10 UDL codes of "Creation of a work center in MRP II" and "Release of a work center in MRP II". 125
7.11 An example of creating and releasing a work center in MRP II. 126

8.1 UPN editor. 154
8.2 UPN first pulldown menu. 156
8.3 UPN second pulldown menu. 158
Chapter 1

Introduction

1.1 Background and Objectives

Due to the substantial improvements in computer technologies and increasing competition in the manufacturing industry, more computerized manufacturing applications have been developed to automate various manufacturing functions in most of the existing factories, and to design new factories. Factory automation started with the introduction of NC machine tools in the early 50s, and continued up to the recent establishment of flexible manufacturing systems and the overall control of the actual flow of products and materials on a factory floor. Current research and computer software developed in the area of manufacturing automation has been quite intensive in dealing with product and process design, production planning, and job execution. However, the design of such systems was made in a functional fashion that emphasized "local" solutions, using closed and self-contained architectures. This, together with the use of heterogeneous databases and incompatible computer operating systems have led to "islands of automation" (figure 1.1) which suffer from data inconsistencies and lack of control of functional interactions between manufacturing application systems. Current and future trends for the use of computers in manufacturing include the control and the integration of information flow of a production operation into a computer-
controlled factory management system. The concept of a factory level Computer Integrated Manufacturing (CIM) system is to fill the gap between the high level production management and the low level factory automation.

![Diagram](image)

Figure 1.1: Islands of automation

The primary objective of our research is to develop a methodology of acquiring domain knowledge (company policy), to verify and implement it as a Knowledge Based System, in order to control the information flow among all those computer based manufacturing application systems. The management and control of information flow is unique in our research, compared to other research whose primary objective is to develop a consistent database framework or a standard communication protocol for data transformation. Our control mechanism, complemented with existing distributed database management systems, can achieve a fully integrated manufacturing information system. Our view of the control in information system is:

**Data ownership**: Each application system has authority to create, update and delete specific data entities, in reflecting specific company policies.

**Precedence constraints**: The pre-conditions of and sequences among activities within all the application systems involved has to be defined according to
specific company policies.

The main objectives to be met by the information system to be developed, are listed below:

- **Elimination of islands of automation:**
  Different computer based manufacturing application systems have been developed and used independently, without communicating with each other. The information flow between them is carried out through unreliable paper work, and it is controlled by various company departments without a unified set of procedural rules [Anderson 84]. Thus, the need for the development of a company-wide policy for the management and control of automated information flow among existing application systems.

- **Incorporation of new applications:**
  New computer based manufacturing application systems are being introduced continuously and these will also need to be integrated into the existing framework [Appleton 84]. Most factories are already using distributed computer systems, with multiple databases, that serve these specific applications in the factory [Jablonski 88]. Thus, the need for modularity.

- **Provision of integrity and security:**
  Data integrity and security in individual databases have become major issues of concern in information integration [Date 90]. Thus, the need for controlled accessibility.

### 1.2 Research Issues

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, can be classified in two categories: Static and Dynamic. The former define the various entities of the distributed system, while the latter deal with the functioning of the system as it operates to satisfy the market demand.

1. Static Data
   
   - Product Data: part master data, part revision records, and bills of material.
   - Resource Data: work centers and labor resources.
   - Process Data: routings or process plans.

2. Dynamic Data
   
   - Planning Data: product demand, satisfied by purchase orders and manufacturing orders.

The functional relationships, which form a shell on top of the integrated system to control the sequences of functions within and among those manufacturing application systems, and the database updates and retrievals, are purely domain dependent and can be very different depending on specific company policies. The methodology, however, for knowledge acquisition, modeling, validation and implementation should be totally generic.

The three major research issues to be addressed are:
• Design of the CIM architecture

The CIM architecture to be developed is concerned with defining the functionalities of each manufacturing application system and the relationships between them.

• Development of a methodology for the design of the KBS

The design and maintenance of a Knowledge Based System (KBS) to control the functional relationships and information flow within the elements of the integrated system is the major task of this research. Based on a graphical modeling tool, Petri Nets, a hierarchical representation of the KBS verifies its properties, and validates the company policy prior to the implementation phase.

• Implementation

A procedure to automatically translate the KBS representation by Petri net models into a rule specification language has been developed, and a prototype test system is in place.

1.3 Contributions of this work

Contributions made in the work of this dissertation are listed as follows:

• Establishment of a design methodology to model, verify and implement knowledge bases, especially for manufacturing information systems, as presented in Chapter 3.

• Development of a new graphical representation for modeling rules and database updates, called Updated Petri Nets (UPN), and an algorithm to translate UPN into a rule specification language, as presented in Chapters 4, 5
and 6. It provides a hierarchical design technique, a database representation, and abstraction techniques for complex models.

- Implementation of a software package to automate the modeling, verification, and implementation of knowledge based systems, using UPN, as presented in Chapter 7.

1.4 Structure of the Dissertation

The second chapter reviews current research related to the development of Computer Integrated Manufacturing systems and related Petri net applications. The third chapter presents our CIM system specification and its architecture, and the fourth chapter details the design methodology of the knowledge base. The fifth chapter describes the Updated Petri Nets (UPN) developed for the purposes of this project and the sixth chapter presents some specific properties of the knowledge base and how inconsistencies and modeling errors can be addressed through analytical methods. The seventh chapter presents the translation of the knowledge base into a specifically designed rule specification language and the overall implementation strategy. The eighth chapter discusses the application of the KBS design methodology and the software package developed to automate it. The last chapter presents our conclusions with recommendations for future work.
Chapter 2

Literature Survey

The literature on CIM systems is becoming vast and greatly diversified. Our survey categorizes the existing body of knowledge in four major sections. The first one describes the architectural issues and some architectures presented in various projects. The second section presents the database work carried out in relation to CIM systems, to satisfy their particular requirements. Applications of expert systems in CIM are outlined in the third section. Finally, in the fourth section we present various modeling techniques that have been proposed to model physical manufacturing systems and related information systems.

2.1 CIM System Architectures and SFC

The National Institute of Standards and Technology (NIST), within its Automated Manufacturing Research Facility (AMRF), has addressed issues related to standardized interfaces for data-sharing, which are important for system integration [Jones 86]. The NIST has developed a manufacturing system architecture, shown in figure 2.1, which is based on a five-level hierarchical structure: Facility, Shop, Cell, Workstation, and Equipment levels of control. Current efforts focus on the design of a real-time production scheduler at the Shop level and the Cell level [Davis 88] [Jones 89], but not yet on the design of a
complete factory integrated system.

Figure 2.1: NIST CIM system architecture.

Project #418 [Bonnevie 87] of the European Strategic Program for Research and development in Information Technology (ESPRIT) has resulted in a hierarchical structure with multi-level dynamic planning and scheduling and a GRAI modeling method to assist decision making. ESPRIT project #932 [Meyer 87] provides an information flow diagram for factory analysis, derived from the hierarchical NIST model, using a SADT modeling method. The research of ESPRIT focuses on the design of a control architecture at Shop and Workcell levels and some results have been already achieved. However, there is no provision for the control of information flow between the factory level and the job shop level.

The Computer Aided Manufacturing - International (CAM-i) has established an Intelligent Manufacturing Management Program (IMMP) to develop the concepts for an advanced factory management and control system. The objectives are to ensure efficient use of resources and to facilitate decision-making at the lowest level and at the last possible moment. So far, one such system has been developed and implemented, known as MADEMA (MAnufacturing DEcision MAking) at
MIT [Chryssoulis 87]. However, this system does not address control and integration of information flow between system modules. Rather, it concentrates on decision making rules for dispatching of work orders to work centers.

PRISM (Productivity Improvement Systems for Manufacturing) [Franks 87], developed in the AT&T Bell Laboratories, presents an integrated architecture of various computerized information systems, supporting manufacturing execution functions, and a useful tool for controlling information flow at AT&T's factories. It provides a logical architecture for the Shop Floor Control system. It concentrates on the scheduling and control of physical Shop Floor activities through its MOVES system, and the interchange of product and process data between its Shop Floor Control system and the other manufacturing application systems (eg. a product and process design system "FOCUS"). Although this system provides the communication between various manufacturing application systems including CAD, CAPP, and SFC, it does not intend to incorporate the control of information flow, based on specific company rules. However, we have adopted their SFC definition and functionality in defining the relationships between SFC and each of the CAD, CAPP, MRP II applications in our work. A detailed description of the functionality of a generic SFC system is provided in section 3.1.2.

2.2 Database Frameworks for CIM

The Integrated Manufacturing Database Administration System (IMDAS) developed by the NIST [Libes 85] [Su 86] emphasizes extending the traditional database management technology, to synchronize the data processing through each module of the CIM system. The application programs communicate with IMDAS through a common interface, using an SQL-like language, referencing data names from a common data dictionary. Depending on the transactions
invoked by the application programs, IMDAS retrieves or updates data through a common interface to the underlying distributed databases. IMDAS is one of the earliest developments of database management systems, interfacing with distributed manufacturing application systems. The IMDAS has influenced the interfacing of our Knowledge Based System with the database management system of each existing manufacturing application.

A manufacturing information management design method was developed at the Rensselaer Polytechnic Institute, [Hsu 87] which uses a two-stage entity relationship (TSER) modeling method and a knowledge-based control methodology in a metadatabase framework. This work provided the design concept and the necessary methodologies to establish a metadatabase for the integration of CAD and MRP systems. However, they have concentrated rather on the development of a database framework for data integrity and consistency, which is also part of our research goal, but not on the development of a knowledge base for controlling the data updates and retrievals based on a given company policy.

The CIM Data Engine (CDE) developed at TRW [Sepehri 87], similar to the concept of IMDAS, translates queries from the application systems and provides interfaces with the databases where the required data resides. The basic function of the CDE is that of a data translator and synchronizing processor for the CIM network. It is trying not to modify the existing application systems, except for an addition of interface software that transmits all transactions and requested data to the CIM engine. Their concept of information integration is very similar to ours, except that, like most of the other CIM systems, it does not address the issue of defining logical rules, which reflect the specific company policy, to control and manage the information flow between different manufacturing application systems.
2.3 Knowledge Based Expert Systems for CIM

A knowledge-based expert system can be the heart of CIM, since it can accommodate a lot of rules and constraints based on manufacturing expertise. Building high-performance knowledge-based expert systems for advanced manufacturing and automation is now the most active research subject within CIM and Artificial Intelligence. The Knowledge-Based Engineering Systems Research Laboratory at University of Illinois-Urbana-Champaign, has developed a framework for knowledge-based computer-integrated manufacturing [Lu 86] which can be used to perform common manufacturing tasks, such as monitoring, diagnostics, control, simulation, and scheduling. Basically, this framework provides a way for manufacturing applications to access data and knowledge residing in the other applications, rather than to define the logical rules that drive the transactions performed for the control of the information flow. The MKS (Manufacturing Knowledge System) at Stanford University [Pan 89] provides a set of tools for modeling a manufacturing environment (including process, equipment, facilities, and operational procedures). However, their approach emphasizes the provision of a single consistent representation of the manufacturing domain that is shared by all applications, in the form of a common database schema. This requires major changes when it is necessary to merge new functions within the existing application systems, as well as major redesigning when incorporating new application systems.

2.4 Systems Modeling Tools

In order to facilitate the design of the information control and management system, it is necessary to use a tool to model and analyze the system during the design stage. In a manufacturing environment, tools are used in modeling not only the physical systems (material processing and flow) but also the interaction of
computer application systems (information flow). Several well known graphic modeling tools are being reviewed in the following sections:

2.4.1 IDEF and SADT

The ICAM Definition (IDEF) developed by the United States Air Force in their Integrated Computer Aided Manufacturing (ICAM) program [Staff 83] was the primary tool of system design and modeling, which includes three modeling techniques: IDEF₀ function model, IDEF₁ information model, and IDEF₂ dynamic (simulation) model. The IDEF₀ and IDEF₁ provide a complete static modeling method for production control and manufacturing systems, and the IDEF₂ is used for dynamic simulation and performance evaluation purposes.

SADT (Systematic Analysis and Design Technique) [Ross 85] is an extended modeling method spun from IDEF. It can be used to define the information flow between all the basic sub-systems within a factory, and is very useful in terms of modeling static data functions and relationships of manufacturing systems. It uses a hierarchical structure and reflects the information flow between each of the functions. However, it can not model logical rules (eg. precedence constraints or the sequences of events) present in a manufacturing environment, because of its limited modeling structure, which has basically two entities: boxes, representing functions or operations, and arrows, representing information exchanges between operations. With only these two entities, it cannot reflect precedence constraints of starting or ending an operation and concurrency of operations. Strictly speaking, SADT can only be used to define the functionalities of each application system and the inputs/outputs between them.
2.4.2 GRAI

The GRAI laboratory at the University of Bordeaux in France has developed the GRAI method [Doumeingts 87] especially for the study of decisional production systems. The GRAI method includes three major parts:

- **Conceptual model**

  The conceptual model of the GRAI method is composed of a Production Control System (PCS) which describes the organization, and a Decision Center (DC) which details various activities. The PCS is split up into three sub-systems: the physical sub-system (machines, men, material flow), the decision sub-system and the informational sub-system linking the previous two sub-systems. The DC defines the exchange of information, activities, and decisions made within a PCS. Both models represent various concepts allowing to develop a consistent, valid and accepted representation. It is based on a set of design and analysis rules.

- **GRAI tools**

  The GRAI method provides two graphic tools: the GRAI grid and GRAI nets. The GRAI grid provides a hierarchical representation of the whole structure of the decision centers in a Production Management System and GRAI nets describe the various activities of each decision center.

- **Structure**

  A specific structure has been developed to apply the GRAI method, using strict procedures which consist of two major phases: the analysis phase which analyzes the current system and collects all data necessary for designing the new system; the design phase which designs the system from data collected
during the previous phase and by analyzing any discrepancies between the current system and the ideal system.

The GRAI method has been developed for analyzing, designing and specifying Production Management Systems in a context of integration. Based on theories of complex systems, hierarchical systems, organization systems, and on the theory of discrete events, it is especially aimed at the study of decisional aspects in manufacturing systems. Therefore, it is not suitable for modeling the dynamic behavior of an information control system.

2.4.3 Petri Nets

On the other hand, Petri nets [Peterson 81] [Reisig 85] [Murata 89] are ideal for modeling dynamically and formally analyzing complex dynamic relationships of interacting systems. They were initially developed and used mainly for advanced computer integrated system design, both for hardware and software [Courvoisier 83], and for flexible manufacturing systems [Crockett 86]. Most recent applications of Petri nets in manufacturing systems are focusing again on the shop floor level, with a large number of work stations, robots, and transportation systems, to be handled by a central controller [Ravichandran 87], [Merabet 87], [Dridi 85]. Research in modeling manufacturing systems has been quite extensive in recent years on system validation and performance evaluation using High Level Petri Nets (HLPN), such as Timed Petri Nets (TPN), Stochastic Timed Petri Nets (STPN), Predicate Transition Nets, Colored Petri Nets (CPN), and Event Graphs, [Alanche 84], [Alla 84], [Kamath 86], [Crockett 87] [Claver 91]. We have chosen to develop a special subset of Colored Petri Nets, which allow the model designer to work at different aggregation levels, in modeling the flow and control of information. The main advantage of CPNs over Generalized Petri Nets (GPN) besides their higher modeling power, is the possibility of obtaining a compact
representation of a large and complex system.

2.5 Conclusion

As a result of this review, it is evident that these efforts, which mainly concentrate on either the operation integration issues at the shop floor level or the data integrity and consistency issues at higher levels, are quite different from our proposed CIM system. We concentrate on controlling the information flow at a higher level—the facility level, based on knowledge expressed in the form of specific company policies. It is, therefore, our belief that our work indeed complements existing CIM architectures. Also, in contrast to single database CIM systems, we propose a database interoperability approach, that has already been used successfully in developing a knowledge-based prototype CAD/CAPP/MRP II integrated system [Mark 87] [Harhalakis 88] [Harhalakis 90], at the Department of Mechanical Engineering and the Systems Research Center of the University of Maryland.
Chapter 3

Research Approach

3.1 Introduction

Knowledge based systems have become a main stream for controlling the information flow and automating the decision processes in a variety of research areas and industrial applications. As seen in the previous chapter, a major application of them is manufacturing automation for the control of both data and physical machining processes. Our research emphasis is placed on the control and integration of information flow of production operations to achieve a computer-controlled factory management system. A similar approach has been taken in [Dilts 91] to develop a framework for integrated CIM database by using knowledge based technology. The system architecture of its integrated CIM databases involves both the distributed database management systems and knowledge based systems for sharing information, and the communication between them is achieved through an integrated interface. Its knowledge base consists of several types of knowledge, including domain knowledge, conceptual data model, logical structure of the database, and data accessibility. Our approach, however, is emphasizing on the knowledge which reflects the company policy, more specifically, the functional relationships of procedures and operations of the involved applications. For example, the creation of an equipment data entity in the database of one application
starts from checking necessary information within the other databases. Upon satisfactory completion of these checking procedures, it will trigger the creation of the same data entity in the database of another application where it will be used. Besides, a formal modeling tool has been developed to model and analyze the knowledge base systematically. The following sections discuss our general research approach that addresses all the related aspects of the system: architecture, design methodology and implementation.

![Diagram showing data flow between CAD, CAPP, MRP-II, and SFC]

Figure 3.1: Data commonalities between CAD, CAPP, MRP II, and SFC

### 3.1.1 CIM System Specification and Architecture

Our integrated system is intended for a discrete-parts, make-to-stock environment, where CAD, CAPP, MRP II and SFC systems are best utilized. As described previously, the design of our model is based on the information flow established between the four functional areas. The common entities involved in this integrated model are Part Master Data (including Part Revisions), Bills of Materials, Work Centers, Process Plans, and Manufacturing Orders. As shown in figure 3.1, these entities are shared by more than one system and there is a constant stream of information related to retrievals, insertions and updates of these entities.
The model does not attempt to provide a custom-made "bridge box", allowing users to hook up any existing commercial CAD, CAPP, MRP II and SFC systems. Our goal is to demonstrate the viability of achieving the integration and the control of information flow, using generic operations on generic entities. In order to remain as general as possible, this model does not emulate any particular CAD, CAPP, MRP II and SFC packages. It relies only on the basic functions available to most such commercial systems. In an actual implementation, however, the system would act as a controller between existing CAD, CAPP, MRP II and SFC packages, while utilizing their respective capabilities. In this case an appropriate model has to be developed, when attempting to control and integrate specific CAD, CAPP, MRP II and SFC software, depending upon their specific features and characteristics, under a specific company policy. Therefore, the value of our work lies mostly in developing a methodology for designing a CIM information system, rather than the system itself.

3.1.2 Role of Each Application System

To maintain the generality of the model, only the most basic data carried by CAD, CAPP, MRP II, and SFC have been incorporated. If necessary, the model can be easily extended to reflect additional data and functions specific to particular commercial packages. Specifying the roles of the respective areas, i.e., CAD, CAPP, MRP II, and SFC is of importance when designing the model.

CAD, being the center of design activity, is the primary controller of product design information. The evaluation of design alternatives, creation of new product parts, and the modification of existing parts is performed within CAD, often using inputs from other departments. Marketing and Manufacturing are two major contributors to information regarding product designs. In addition, CAD initiates the bills of materials for all product assemblies. An important problem commonly
encountered is that, as a function, manufacturing succeeds design. Therefore any manufacturing problems occurring due to part design specification, are relayed to CAD only after designs have been finalized. It is, therefore, necessary for CAPP, the originator of process plans in the system, to work in concert with CAD, as the design of a part is ongoing. This approach, known as concurrent engineering, reduces the product development cycle and enhances competitiveness.

CAPP is solely responsible for developing manufacturing process plans. It organizes the manufacturing activities to be performed on a part, into specific operations, each being assigned to a particular workcenter, and each requiring tools, jigs, fixtures and set up and run times. CAPP on the other hand can initiate its own parts and bills of materials as they relate to necessary tools, jigs, and fixtures for production purposes. In addition, most CAPP systems maintain detailed workcenter files, the information being used while preparing process plans.

MRP II plays a coordinating and monitoring role. It plans for and monitors the actual procurement of raw materials and manufacture of parts, respectively. It can also initiate non-product parts such as supply items into the system. In addition, it records process plans as generated by CAPP, and also product structures of assembly parts, to provide them later to the SFC module. Workcenter data are maintained here, with MRP II having sole discretion as to their initiation and deletion in the system.

While the definitions and functionalities of CAD, CAPP, and MRP II systems are quite clear and widely accepted, the functions and inputs/outputs of a Shop Floor Control system in a CIM environment are not yet well defined. Shop Floor Control is basically a system which directly controls the transformation of planned manufacturing orders into a set of jobs, and the transformation of raw materials into products. The basic activities of a Shop Floor Control module are listed below [Melnyk 85]:

19
• Capacity planning and resource allocation based on inputs from MRP II.

• Short-term capacity adjusting by using alternative routings, planning overtime, etc.

• Feedback for reporting machine performance and status, job completion stage, and actual labor and material usage.

![Diagram of Shop Floor Control System]

Figure 3.2: Definition of a generic shop floor control system.

Our definition of a generic Shop Floor Control system and identification of the input/output requirements, has been influenced by a study of Shop Floor Control systems which have been used by three major research projects: NIST[Jones 86], ESPRIT[Bonnevie 87], MADEMA[Chryssolouris 87]. The Productivity Improvement Systems for Manufacturing (PRISM) [Franks 87] developed in AT&T Bell Laboratories has been our primary guideline. The basic
functions and inputs/outputs of a generic Shop Floor Control system are shown in figure 3.2.

SFC is designed to communicate with an MRP II Planning system and to perform job scheduling and monitoring using detail routing information from CAPP. Once a market demand arrives, MRP II will generate planned purchasing and manufacturing orders. In doing this, lead times from Part Master Records and existing inventories are taken into account. SFC will then schedule the manufacturing jobs, based on the current load and the detail routing information from CAPP. It will then dispatch these job orders down to the shop floor. During production, SFC will constantly monitor the job status, work center status, actual material and labor consumption, and report them back to the MRP II for costing and updating purposes. Finally SFC is supposed to react to and provide real time solutions in the event of disturbances, such as machine breakdowns, critical labor absenteeism and material shortages. Issues that can not be resolved at the SFC level must be communicated to MRP II for further action, [Nagi 88].

3.1.3 Multi-Database versus Central Database

The basic approach of information integration used in our research is to build a knowledge based system to control an integrated information flow between multiple databases of all existing manufacturing applications, and to cater for the addition of new applications as they emerge. Alternative approaches include:

- Design and installation of a complete new, fully integrated system [Lu 86]

- Design of a single centralized data base scheme to be accessed by all existing application systems [Hsu 87].

The first approach involves substantial investment in replacing computer hardware and software, and retraining of all related personnel. The second
approach requires that all the data of the application systems being integrated to be converted into a common database schema and all the application systems to be rewritten, which requires substantial effort and still limits the incorporation of new computer based manufacturing applications. These two alternative approaches may apply to small factories with less amount of data to be processed and less sophisticated systems to operate. Our approach, on the other hand, uses the existing application systems and tries to reduce the amount of modification to them. It is also capable of incorporating new applications in a modular fashion without the need to re-design the entire system.

![Diagram](image)

Figure 3.3: Information flow architecture for manufacturing application systems

### 3.1.4 Overall CIM Information Flow Architecture

Our CIM architecture concentrates on the integration of manufacturing applications at the Factory level (similar to the Facility level of the NIST hierarchy) as depicted in figure 3.3. CAD, CAPP, MRP II, and SFC’ can be integrated together through a general Distributed Database Management System (DDBMS). The Knowledge Based System which is the subject of our research drives the DDBMS to control the information flow, following procedural rules, constraints and other guidelines derived from the company policy. In order to build a prototype of
the CAD/CAPP/MRP II/SFC integrated system, we have defined data structures of the common data entities involved in the various manufacturing applications of our integrated system and their relations, which are stored in the DDBMS. Therefore, it can be said that the management and control of information flow is performed by the KBS, while the integration aspect is addressed by the DDBMS.

3.2 System Design Methodology

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 of this research. Our general design methodology for it is depicted in figure 3.4.

![Figure 3.4: Knowledge base design methodology](image)

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 Nets) and a hierarchical modeling methodology, discussed in chapter 4. The next step is to convert the UPN model into General Petri Nets (GPN) for validation purposes, and feed the results back to the user to resolve (i)
conflicting company rules and (ii) errors introduced during the modeling phase. This process is described in chapter 5. After the model has been validated, a parser translates the UPN model into a rule specification language, described in chapter 6. The end result is a software package that controls the data-flow and accessibility between several databases. In short, the input is a specific company policy, expressed as a set of logical and procedural rules, and the output is an AI production system for controlling operations, accessibility, precedence, and updates of data within the manufacturing applications involved. An overview of the major design phases of the system is presented below.

3.2.1 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. As mentioned above, the expert rules embedded in the knowledge base are extracted from company expertise, policies and procedures, 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 base. However, since we are here to develop and demonstrate a generic 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.

Scenarios of the Proposed CIM System

The development of a knowledge based system usually starts from designing a set of abstract rules for the initiation and maintenance of specific entities within the system. A set of scenarios under each entity which represent these abstract rules
is listed below.

Part Data

- Adding New Product Parts in CAD
- Adding New Product Part Revisions in CAD
- Adding New Non-Product Parts in CAPP
- Adding New Non-Product Part Revisions in CAPP
- Adding New Non-Product Parts in MRP II
- Adding New Non-Product Revisions in MRP II
- Making Parts Obsolete
- Deleting Parts

Product Structures (Bills of Material)

- Adding Component Relationships in CAD (for products)
- Adding Component Relationships in CAPP (for tools, jigs, fixtures etc.)
- Deleting Component Relationships
- Substituting Components in Relationships
- Changing the Required Quantity of a Component
- Copying Relationships from One Assembly to Another

Work Centers

- Establishing New Work Centers in MRP II
• Modifying Work Center in MRP II, CAPP, and SFC

• Deleting Work Centers in MRP II

**Process Plans**

• Establishing New Process Plans in CAPP

• Modifying Process Plans in CAPP

• Deleting Process Plans in CAPP

**Manufacturing Orders**

• Adding Orders in MRP II

• Modifying Orders in MRP II

• Modifying Order BOM/Routing in SFC

• Updating Job Status in SFC (including actual material issued and time taken)

• Deleting Orders from MRP II

**Status Codes**

The flow of information within the system is controlled using a set of status codes assigned to each set of entity data within each functional area. The status codes are designed to provide for triggering the right action while controlling the sequence of various part and process design and manufacturing related activities. The following are the status codes used in the system.
**Working** : The "working" status is given to CAD part data related to designs that have not yet been finalized. In a similar fashion, it is given to process plans in CAPP, work centers in CAPP, and shop orders in SFC. In the case of workcenters, it is intended to signify that some work center data is still missing in CAPP.

**Released** : The "released" status is indicative of an entity becoming active in the system. It is applied to CAD and MRP II part revision data; CAPP, SFC, and MRP II work center data; CAPP and MRP II process plan data; SFC and MRP II order data.

**Hold** : The "hold" status is normally given to an entity, when it is being reviewed for possible revision or replacement, and the entity should not be used while on hold. For example, it can be given to a work center in the case of an extended breakdown. It is used for CAD and MRP II part revision data; CAPP, SFC, and MRP II work center data; CAPP and MRP II process plan data; SFC and MRP II order data.

**Obsolete** : Data related to entities that are no longer considered active, are given the "obsolete" status. This code is used by CAD part revisions and CAPP routings. However, MRP II and SFC handle obsolescence with the use of effectivity start and effectivity end dates. Therefore they do not require this status code.

### 3.2.2 Modeling of the Domain Knowledge

Although general Petri nets could initially handle our system modeling, it became necessary to define more complex semantics for nets, in order to handle the increasing complexity of the domain knowledge, due to the involvement of more applications and their entities, and the growing size of it. Hence we have developed
the Updated Petri Nets (UPN), and a hierarchical modeling approach. UPN are defined and presented in chapter 4. The hierarchical modeling approach facilitates the modeling task. It incorporates: (a) a top-down stepwise refinement technique for the modeling of each scenario from an abstract and aggregate level to a detailed level and (b) a synthesis technique for synthesizing separate nets, which represent different scenarios of the system, to form a coherent net. The approach necessitates the development of new Petri nets modeling entities which include two types of transitions; one to represent primitive operations, and the other to represent compound operations which can be further exploded into sub-nets. The design process for each scenario (each set of functional relations) starts from an abstract net with both primitive and compound transitions, and continues by exploding each compound transition until no compound transition exists. This improved modeling capability enables the structuring of Petri nets in a progressive manner, and facilitates the transformation of the "company policy" of figure 3.4 to a formal model.

3.2.3 Knowledge Verification

Several analysis techniques for Petri nets, including reachability trees and net invariants, are used in this work [Peterson 81] [Jensen 86]. 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 deadlock or inconsistencies in the model. (see chapter 5) The programs for computerizing these analysis methods have been developed and applied extensively. Some reduction rules [Lee 85] have also been investigated for reducing the complexity of nets for further analysis.

However, these analysis techniques were developed for Generalized Petri Nets
(GPN), but not applied to Colored Petri Nets (CPN) in which a great diversity of linear functions are associated to the arcs. Therefore, unlike analysis algorithms for GPN that use integer matrices, analysis algorithms for CPN need to manipulate matrices composed by linear functions. This fact introduces a high complexity in the development and execution of these algorithms. An alternative approach is to convert UPN into GPN before they can be analyzed. The unfolding algorithm has been developed and presented in chapter 5.

3.3 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 [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 and retrieval dependencies. These update and retrieval dependencies control inter-database consistency through inter-database operation calls. We propose the use of Update Dependencies as a special rule specification language, to be used for the implementation of our Knowledge Based System. The algorithm of automatic translation between the UPN and the UDL has been developed and presented in chapter 6, which reduces dramatically the implementation effort. The details of the Update Dependencies are also presented in the same chapter.
Chapter 4

Updated Petri Nets (UPN) for Information Systems

4.1 Introduction

The emphasis in this chapter is on developing a powerful representation tool which can be analyzed, in order to validate the underlying domain knowledge extracted from the company policy, and that can be implemented into rule production systems automatically.

Petri Nets were originally developed by Carl Adam Petri in his doctoral thesis, 1962, at the University of Darmstadt, West Germany. There have been many reports and papers published on Petri Nets with a wide variety of applications due to their generality and modeling power. Petri Nets can be applied to most systems in representing graphically not only sequential but also concurrent activities. 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. Readers may refer to Peterson's book [Peterson 81] for the fundamentals of Petri Nets theory. A survey of literature
where various types of Petri Nets are used in modeling various systems in general and manufacturing systems in particular, has been presented in chapter 2, section 2.4.

4.2 Evolution of Updated Petri Nets

As explained in chapter 3, section 3.2.2, it became necessary to develop the Updated Petri Nets (UPN), a special type of the Colored Petri Nets (CPN) [Jensen 87], and a hierarchical modeling methodology with a systematic approach for the synthesis of separate nets with common places. 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. We use UPN in modeling not only the rule base, but also the database changes which ensure consistency in representing database statuses.

In addition, due to the nature of Petri nets, UPN maintain the capability of modeling concurrent execution of transitions which are enabled simultaneously. When operating a manufacturing information system, more than one procedures can be executed at the same time. For example, when a MRP II user is creating a work center record, another MRP II user may be making a deletion of one of the other existing work centers. This control of concurrent execution is usually part of the query strategy of most database management systems, which can also be modeled by UPN.

4.3 Updated Petri Nets (UPN) Formalism

This section describes the formalism for knowledge representation of an information system modeled by UPN. We have extended the primitives of the general Colored Petri net descriptions in order to reflect, more closely, the terminology and
semantics involved in the database application domain. These primitives are used to develop a procedure to automate and formalize the interpretation process from the model to the rule specification language. In the following paragraphs we present the formal definition of UPN, which is based on both the CP-graph definition and CP-matrix definition given by [Jensen 87].

An UPN is a directed graph with three types of nodes: places which represent facts or predicates, primitive transitions which represent rules or implications, 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: \( UPN = \langle P, T, C, I^-, I^+, M_0, I_o, MT \rangle \), which can be decomposed in four different parts:

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 is used in the validation process. Its entities are defined as follows [Jensen 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).
   - \( P \cap T = \emptyset \) and \( P \cup T \neq \emptyset \).
   - \( C \) is the color function defined from \( P \cup T \) into non-empty sets. It attaches to each place a set of possible token-data and to each transition a set of possible data occurrences.
   - \( I^- \) and \( I^+ \) are negative and positive incidence functions defined on \( P \times T \), such that \( I^-(p, t), I^+(p, t) \in [C(t)_{MS} \rightarrow C(p)_{MS}] \) \( \forall(p, t) \in P \times T \),

32
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 \([\ldots]_L\) denotes a set of linear functions\(^1\).

- The net has no isolated places or transitions:
  \[
  \forall c \in P, \exists t \in T : I^-(p, t) \neq 0 \vee I^+(p, t) \neq 0 \quad \text{and} \quad \forall t \in T, \exists p \in P : I^-(p, t) \neq 0 \vee I^+(p, t) \neq 0
  \]

- \( M_0 \) the initial marking, a function defined on \( P \), 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:
  \( I_o(p, t) \in [C(t)]_{MS} \rightarrow [C(p)]_{MS}_L, \forall (p, t) \in P \times T. \)

3. \( MT = \{hm_1, \ldots, hm_l\} \) denotes the set of related transition sets. These are sets of transitions grouped into subnets.

From a rule based perspective, we have divided the representation of knowledge base components in the following four groups: Data, Facts, Rules, Metarules. In the following sections we define the syntax and semantics of the UPN elements representing those components. To illustrate these aspects, we focus on an example scenario, \emph{creation of a work center via MRP II} and, particularly, in one of its subnets, the \emph{release of a work center in MRP II} as shown in figure 4.1 with its initial marking, which is used all through this section. The specification of the example is expressed in natural language (appendix A) and then represented here using UPN primitives.

\(^1\)Although, any linear function is allowed in the general color Petri net, only projections, identities and decolorizing functions have been needed so far in our models.
Figure 4.1: Subnet of the work center creation scenario: "Release of a work center in MRP II" with initial marking.
4.3.1 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, called database relations or records. In order to: (a) provide a more adequate representation to facilitate the translation of the user specifications, (b) give a more clear graphical representation and (c) make the validation process easier, UPN allow for the specification of the following classes of information (in general we refer to any of these data classes by the generic name "data" or "color");

**Atomic data** is individual information with a lexical representation. Each atomic data is represented by an identifier, a color, which is attached to a specific token, whose syntax is an alpha-numerical sequence. As an example, let us suppose that a part in CAD can be in four different statuses: \( w \) (working), \( r \) (released), \( h \) (hold), or \( o \) (obsolete). We can describe them as four constant symbols \( \{w, r, h, o\} \) which can be grouped to form the status color set: \( DSTS = \{w, r, h, o\} \).

Atomic data can belong to one of the following two classes: **fixed**, if the color set is completely known in advance, such as the set status which was described above \( DSTS = \{w, r, h, o\} \), and **non-fixed**, if not all of the color set components are completely known in advance, such as the information about a part identification number in CAD, which can be an alphanumerical sequence (e.g. C157635) or the description of a part record which can be any string (e.g. "Fine-pitch involute spur gear").

This last example raises another interesting point related to non-fixed data. There is some information, such as a part identification number, that is normally used to guide the information control; generally, preconditions of a
rule can be related to a specific part identification number, but normally, specifications do not care about the specific instances of its description; in other words, the description is associated to a specific part and it will be flowing with it in the system, but no precondition check for a specific description is necessary. Thus, non conditional data refer to information which is not important for the control of the system and their flow is controlled by conditional data.

**Data structure** is the classical mechanism to aggregate related data. Therefore, instances of data structures can be seen as ordered tuples \(< s_1, s_2, \ldots, s_n \>\). Different tuples can be identified by their relation name, \(< R >\), and different elements of a tuple can also have textual identifications \(A_1, \ldots, A_m\). This means that every tuple will be represented as \(< R > (A_1, \ldots, A_m)\), where \(< R >\) is the database relation name and \(A_1, \ldots, A_m\) are textual identifications for its attributes. A specific element of a tuple can be accessed in two ways; either by referring its specific position within that tuple ("\(-, -, \ldots, v_i, \ldots, -\)"), or by referring its attribute name ("\(A_i = v_i\)").

To illustrate the model adopted for data representation, let us consider the specifications related to a work center record in MRP II. This record is represented in UPN by a data structure named \(Mwc\) (where \(M\) identifies the MRP II database and \(wc\) identifies the work center record). Its composition, characteristics of its attributes, and its representation are shown in table 4.1.

### 4.3.2 Facts

Fundamental in describing an information system are the facts within it. Each fact declares a piece of information about some data, or data structure, in the system. To be able to model the access and exchange of information in a consistent and
### Complete data structure for work center in MRP II

Complete data structure for work center in MRP II

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data class</th>
<th>Color set</th>
<th>DB data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcid</td>
<td>non fixed</td>
<td>WCID</td>
<td>identification</td>
<td>identification number</td>
</tr>
<tr>
<td>des</td>
<td>non fixed</td>
<td>DES</td>
<td>text</td>
<td>description</td>
</tr>
<tr>
<td>dep</td>
<td>non fixed</td>
<td>DEP</td>
<td>text</td>
<td>department</td>
</tr>
<tr>
<td>cap</td>
<td>non fixed</td>
<td>CAP</td>
<td>integer</td>
<td>capacity</td>
</tr>
<tr>
<td>sts</td>
<td>fixed</td>
<td>MSTS</td>
<td>{h, r} (hold, release)</td>
<td>work center status code</td>
</tr>
<tr>
<td>ste</td>
<td>fixed</td>
<td>MSTE</td>
<td>{na, av} (not avail., avail.)</td>
<td>work center state code</td>
</tr>
<tr>
<td>res</td>
<td>non fixed</td>
<td>RES</td>
<td>text</td>
<td>resource code</td>
</tr>
<tr>
<td>esd</td>
<td>non fixed</td>
<td>ESD</td>
<td>date</td>
<td>effectivity start date</td>
</tr>
</tbody>
</table>

### Complete data structure for work center in CAPP

Complete data structure for work center in CAPP

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data class</th>
<th>Color set</th>
<th>DB data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcid</td>
<td>non fixed</td>
<td>WCID</td>
<td>identification</td>
<td>identification number</td>
</tr>
<tr>
<td>des</td>
<td>non fixed</td>
<td>DES</td>
<td>text</td>
<td>description</td>
</tr>
<tr>
<td>dep</td>
<td>non fixed</td>
<td>DEP</td>
<td>text</td>
<td>department</td>
</tr>
<tr>
<td>cap</td>
<td>non fixed</td>
<td>CAP</td>
<td>integer</td>
<td>capacity</td>
</tr>
<tr>
<td>sts</td>
<td>fixed</td>
<td>PSTS</td>
<td>{w, h, r} (working, hold, release)</td>
<td>work center status code</td>
</tr>
</tbody>
</table>

### Table 4.1: Data structures and color sets of work centers.

generic way, we need to define specific semantics to describe facts. This becomes more important if we consider a modular modeling methodology, where the user must be allowed to create different scenarios which can be linked together later, based upon shared knowledge.

Facts in UPN are represented by places. Places are one of three kinds of nodes in an UPN and are graphically represented as circles. The fact asserted by one place is determined by the place name and its content. This content defines the marking of the place; we will refer to the marking of place \( p \) by \( M(p) \). For example, the fact that a token of color \( A \) is in place \( \text{inprocess} \), can be interpreted as: "\( A \) is in process" or \( \text{inprocess}(A) \) in logic syntax.

Places \( N M wc \) and \( N P wc \) in figure 4.1 represent the non existence of a record in MRP II and CAPP databases respectively. On the other hand, places \( E M wc \) and \( E P wc \) represent the existence of a record. The interpretation associated with each of the rest of the places is provided in figure 4.1.

37
Due to the modular approach, UPN allow the splitting of the whole system into separate scenarios represented by subnets. Each subnet can be designed and verified separately from each other, and then the user can follow a step-wise process for merging different subnets. Regarding facts, places can be of two different types:

**Local scope**: when a place is used to represent a local fact (relevant only to one specific scenario or subnet) which will not affect the state of the other scenarios or subnets. They are typically used to represent intermediate state facts within the decision process. Examples of local places are \( p_1, p_2, p_3, p_4, p_5 \) in figure 4.1.

**Global scope**: when a place is used to represent a fact relevant to (accessible by) different scenarios or subnets. Global places must be referred by the same name in all of their occurrences. Typical examples of global places are facts about database state information, such as places \( EM_{wc} \) and \( NM_{wc} \) (in the MRP II database) and \( EP_{wc} \) and \( NP_{wc} \) (in the CAPP database) shown in figure 4.1.

Regarding the user interface, there are two special types of places to specify the interchange of information with the user (these places are of local scope):

**Input** places represent facts whose initialization is generated by an external action (i.e. by a request from the user through the interface). An example of an input place is the place \( p_1 \) shown in figure 4.1, which needs the introduction of a token for the release process to start.

**Output** places are used to represent situations where some information must exit (i.e. some information to be displayed). An example of an output place is the place \( p_4 \) shown in figure 4.1; the value provided in \( wcid\# \) will be displayed.
The concept of input and output places is similar to that of source and sink places [Beck 86] and, in addition, provides a mechanism to specify terminal facts during consistency and completeness verifications of the knowledge model.

4.3.3 Rule specification

Another important feature in modeling information systems is the representation of information flow specifications. Here, we are considering domains where the user specifies information flow policies using “if then” rules. These rules are expressed in UPN by means of other type of net nodes called transitions. Any transition $t$ has a color set, $C(t)$, associated with it. The net in figure 4.1 shows five transitions $(t_1, t_2, t_3, t_4, t_5)$ and their associated interpretations.

In this section we explain how several components of the rules (preconditions, postconditions, variables, etc.) are represented and used in UPN.

Variables

UPN characterize references to generic data using variables (in the sense that a relation can hold for different data occurrences). There exists a set $V$ of typed variables. Each variable $v : D$ has a name $v$ and an associated color set $D$. They are represented graphically by names ending with the symbol “#”. Examples of variables are $\text{wcid#}, \text{des#}, \text{dep#}$, and $\text{cap#}$ as shown in figure 4.1.

Arcs

Arcs are relations which connect facts and actions to form rules, and constitute the rule preconditions and postconditions. Arcs also identify the flow of information. Formally, each arc has attached to it an arc expression, $exp$, containing a set of variables or constants $\{v_1 : D_1, v_2 : D_2, \ldots, v_n : D_n\}$, where $D_i$ identifies the color
set of the variable or constant \( v_i \).

The lambda-expression, \( \lambda(v_1, v_2, \cdots, v_n).exp \) is defined as a mapping from \( D_1 \times D_2 \times \cdots \times D_n \) into \( C(p) \), where the place \( p \) is the source/destination of the arc.

UPN use variables as in typical rule based systems, where they are used to specify data sets within the enabling arcs of a rule and as a mechanism to transfer data from the preconditions to the postconditions. For example, looking at transition \( t_4 \) in figure 4.1, the information about variables \( des\# \) and \( dep\# \) are needed for place \( p_5 \), which will be transferred from place \( EMwc \), as shown in their arc expressions.

In general in colored Petri nets, upon the firing of a transition, the enabled colors are taken out from the incoming places. We have introduced a special case in UPN, in situations where the place represents a database related fact and the color represents a record. In case of firing, all the information from that record is removed from the place, while only a subset of its attributes may be involved for the update of information. The information of those unspecified attributes must also be included in the color set of the transition, in order to transfer it from one database to the other or back to itself. For example, looking at transition \( t_5 \) in figure 4.1, the attributes \( des, dep, \) and \( ste \) of the MRP II work center table are not shown in the arc expressions for the update of the work center record (pair of arrows from and to the place \( EMwc \)). However, these attributes have to be included in the colored set of transition \( t_5 \) as shown in table 4.1. One of these variables is used for each record attribute, which is not specified in the expressions of arcs connected to that transition. They are identified by names ending with the symbol "0".

Each transition can have an associated predicate, \( pred \), to impose additional constraints in the enabling of the transition. This predicate is a boolean expression
Figure 4.2: UPN representation of the rule "Release work center from MRP II".
which can only contain those variables which are already in the expressions of all arcs connected to the transition. To avoid degenerate transitions, the predicate must differ from the constant predicate false. The predicate is supposed to be true by default. Simple predicates can be grouped by means of the following logical operators: \(\neg\) (not), \(\lor\) (or) and \(\land\) (and).

The color set, \(C(t)\), of a transition, \(t, \forall t \in T\), is determined by [Jensen 87]:

\[
C(t) = \{(d_1, d_2, \ldots, d_n) \in D_1 \times D_2 \times \cdots \times D_n \mid (\lambda(v_1, v_2, \ldots, v_n).pred)(d_1, d_2, \ldots, d_n)\}
\]

where:

- \(pred\) is the predicate attached to \(t\) and
- \(V(t) = \{v_1 : D_1, v_2 : D_2, \cdots, v_n : D_n\}\) is the set of all variables appearing in the expressions of all arcs connected to the transition.

As an example, let us look at figure 4.2a which focuses on transition \(t_5\) from the net shown in figure 4.1. The color sets for the associated places are as follows (the simple color sets are specified in table 4.1):

\[
C(EMwc) = MWC = WCID \times DES \times DEP \times CAP \times MSTS \times MSTE \times RES \times ESD
\]

\[
C(NPwc) = WCID
\]

\[
C(EPwc) = PWC = WCID \times DES \times DEP \times PSTS
\]

\[
C(ps) = WDDCS = WCID \times DES \times DEP \times CAP
\]

There are four variables in the arcs connected to \(t_5\) (\(wcid\#, des\#, dep\#\) and \(cap\#\)). On the other hand, \(t_5\) has an incoming arc from place \(EMwc\), a database place for the record \(Mwc(wcid, des, dep, cap, sts, ste, res, esd)\), and there are five attributes which do not need to be specified in the arcs to/from \(EMwc\): \(des, dep, ste, res\) and \(esd\). The total set of variables and the color set for the transition \(t_5\) are:
\[ V(t_5) = (\text{wcid}#, \text{des}0 : \text{DES}, \text{dep}0 : \text{DEP}, \text{cap}# : \text{CAP}, \text{mste}0 : \text{MSTE}, \text{res}0 : \text{RES}, \text{esd}0 : \text{ESD}, \text{des}# : \text{DES}, \text{dep}# : \text{DEP}) \]

\[ C(t_5) = \text{MWCSDD} = \text{WCID} \times \text{DES} \times \text{DEP} \times \text{CAP} \times \text{MSTE} \times \text{RES} \times \text{ESD} \times \text{DES} \times \text{DEP} \]

Functions in \( I^- \) and \( I^+ \) are defined in terms of lambda expressions with the form
\[ f(c) = \lambda(V).\text{exp}(c), \text{ where } c \in C(t) \text{ and } \text{exp} \text{ is the expression associated to the arc.} \]

For transition \( t_5 \), \( V \) and \( c \in \text{MWCSDD} \) can be represented as follows:

\[ V = (\text{wcid}#, \text{des}0, \text{dep}0, \text{cap}#, \text{mste}0, \text{res}0, \text{esd}0, \text{des}#, \text{dep}#) \]

\[ c = (\text{wcid}, \text{des}0, \text{dep}0, \text{cap}, \text{mste}0, \text{res}0, \text{esd}0, \text{des}, \text{dep}) \]

UPN provide different types of arcs:

**Enabling arcs** are directed arcs which connect a place/fact with a transition/action and define a precondition for the transition. They indicate the data that must be in a place for a transition to be enabled and must be removed from that place on firing. Given \( C(t) \) and \( V(t) \) defined as above for some transition \( t \in T \), if a single arc from a given \( p \in P \) exists, with an arc expression \( \text{exp} \), \( I^-(p, t) \) is then defined to be the function \( \lambda(v_1, v_2, \ldots, v_n).\text{exp} \) (mapping between \( C(t) \) and \( C(p) \)). If no arc from \( p \) to \( t \) exists, \( I^-(p, t) \) is the zero-function. For example, the following are the enabling arcs for transition \( t_5 \) in figure 4.2a:

- \( I^-(p_5, t_5): \text{exp} = \begin{bmatrix} \text{wcid}# \\ \text{des}# \\ \text{dep}# \\ \text{cap}# \end{bmatrix}, \) 
  \[ \lambda(V).\text{exp} \in [\text{MWCSDD}_{MS} \rightarrow \text{WDDCS}_{MS}]_L \text{ such that:} \]
  \[ \lambda(V).\text{exp}(c) = (\text{wcid}, \text{des}, \text{dep}, \text{cap}) \]

- \( I^-(EMwc, t_5): \text{exp} = [wcid = \text{wcid}#], \) 
  \[ \lambda(V).\text{exp} \in [\text{MWCSDD}_{MS} \rightarrow \text{MWC}_{MS}]_L \text{ such that:} \]
\[ \lambda(V).exp(c) = (wcid, despo, depo, -, -, msteo, reso, esdo) \]

- \( I^-(NPwc, t_5) : \exp = \left[ \begin{array}{c} wcid\# \\ des \end{array} \right], \)

\[ \lambda(V).exp \in [MWCSDD_{MS} \rightarrow WCID_{MS}]_L \text{ such that:} \]

\[ \lambda(V).exp(c) = wcid \]

**Causal arcs** are directed arcs which connect a transition with a place and define a postcondition for the rule. Causal arcs describe modifications to be performed in the state of the net when the transition is fired; and more concretely, they indicate which data must be added to a place after the transition has been fired. \( I^+ \) is defined in the same way as \( I^- \), but by means of the arcs from transitions to places. For example, these are the causal arcs for transition \( t_5 \) in figure 4.2a:

- \( I^+(EMwc, t_5) : \exp = \left[ \begin{array}{c} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ cap = cap\# \\ sts = r \end{array} \right], \)

\[ \lambda(V).exp \in [MWCSDD_{MS} \rightarrow MW_{MS}]_L \text{ such that:} \]

\[ \lambda(V).exp(c) = (wcid, deso, depo, cap, r, msteo, reso, esdo) \]

- \( I^+(EPwc, t_5) : \exp = \left[ \begin{array}{c} wcid = wcid\# \\ des = des\# \\ dep = dep\# \\ sts = w \end{array} \right], \)

\[ \lambda(V).exp \in [MWCSDD_{MS} \rightarrow PW_{MS}]_L \text{ such that:} \]

\[ \lambda(V).exp(c) = (wcid, des, dep, cap, w) \]

**Checking arcs** indicate which data must be present in a place, in order to enable a transition, but no data are removed upon firing. They are represented by an enabling and a causal arc, with a unique arc expression. These are generally used to indicate database checkings. For example, transition \( t_2 \) and place
\( EMwc \) in figure 4.1 are connected by a checking arc with the associated arc expression: \( Mwc(\text{wcid} = \text{wcid#}) \). This is a precondition for \( t_2 \) to be enabled, but after the transition has been fired, no data are removed from or added to \( EMwc \). Arcs between \( t_2 \) and \( NMwc \) and arcs between \( t_4 \) and \( EMwc \) are also checking arcs.

**Inhibitor arcs** indicate which data must not be present in a place in order to enable a transition. It is graphically represented as an enabling arc ending in a small circle.

It is also possible for a transition to request information from the user interface. This is done whenever an arc expression requires some information that has not been provided by the variables in the enabling or checking arcs.

**Transition enabling and firing**

The dynamic behavior of the net is provided by transition enabling and firing. A transition \( t \in T \) is said to be enabled with respect to a color \( c \in C(t) \) if the current marking \( M \) is such that:

\[
M(p) \geq I^-(p,t)(c) \quad \text{and} \quad M(p) < I^+(p,t)(c), \quad \forall p \in P
\]

An enabled transition may be chosen to fire. The firing of a transition \( t \) with respect to a color \( c \) consists of removing \( I^-(p,t)(c) \) colors from each of its input places and adding \( I^+(p,t)(c) \) colors to each of its output places. We may think of a firing as an event which may take place if certain conditions are satisfied. Each firing creates a new set of conditions, and the total number of colors in the net may change after each firing.

In order to illustrate the previous concepts, we focus on transition \( t_5 \) in figure 4.2a. For this explanation, let us consider the following initial marking:
\[ M_0(N M w c) = \emptyset \]

\[ M_0(E M w c) = \langle vmc12, \text{"Fadal VMC"}, \text{"machine shop"}, \text{null}, w, na, m12, \text{null} \rangle = M w c(\text{ucid} = \text{vmc12}, \text{des} = \text{"Fadal VMC"}, \text{dep} = \text{"machine shop"}, \text{cap} = \text{null}, \text{sts} = w, \text{ste} = na, \text{res} = m12, \text{csd} = \text{null}) \]

\[ M_0(N P w c) = \text{vmc12} \]

\[ M_0(E P w c) = \emptyset \]

\[ M_0(p_5) = \langle \text{vmc12, "Fadal VMC"}, \text{"machine shop"}, 8 \rangle \]

It is clear that transition \( t_5 \) is enabled with respect to the following specific color (note that the entire color set for transition \( t_5 \) is \( C(t_5) = M W C S D D \)):

\[ c = \langle \text{vmc12, "Fadal VMC"}, \text{"machine shop"}, \text{null}, na, m12, \text{null, "Fadal VMC", "machine shop"} \rangle \]

Because \( c \) satisfies the following enabling rules, \( t_5 \) is then enabled.

\[ M_0(N M w c) \geq \lambda(V)exp(c) = \emptyset \]

\[ M_0(E M w c) \geq \lambda(V)exp(c) = \text{vmc12, "Fadal VMC"}, \text{"machine shop"}, \text{null}, w, na, m12, \text{null} \]

\[ M_0(N P w c) \geq \lambda(V)exp(c) = \text{vmc12} \]

\[ M_0(E P w c) \geq \lambda(V)exp(c) = \emptyset \]

\[ M_0(p_5) \geq \lambda(V)exp(c) = \langle \text{vmc12, "Fadal VMC"}, \text{"machine shop"}, 8 \rangle \]

Where:

\[ exp \text{ represents the corresponding arc expression,} \]

\[ V \text{ is as defined in section 4.3.3} \]

The firing of transition \( t_5 \), with respect to color \( c \), implies the removal of the tokens corresponding to the arc expressions from the input places and the addition of the tokens defined by the causal arcs to the output places:
\[ M_0(NMwc) = \lambda(V)\exp(c) = \emptyset \]
\[ M_0(EMwc) = \lambda(V)\exp(c) = (vmc12, "Fadal VMC", "machine shop", 8, r, na, m12, null), \]
\[ M_0(NPwc) = \lambda(V)\exp(c) = \emptyset \]
\[ M_0(EPwc) = \lambda(V)\exp(c) = (vmc12, "Fadal VMC", "machine shop", 8, w) \]
\[ M_0(p_8) = \lambda(V)\exp(c) = \emptyset \]

The final marking after transition \( t_5 \) fires is the following (figure 4.2b):

\[ M_0(NMwc) = \emptyset \]
\[ M_0(EMwc) = (vmc12, "Fadal VMC", "machine shop", 8, r, na, m12, null), \]
\[ M_0(NPwc) = \emptyset \]
\[ M_0(EPwc) = (vmc12, "Fadal VMC", "machine shop", 8, w) \]
\[ M_0(p_8) = \emptyset \]

### 4.3.4 Metarule specification

Metarules provide a higher level representation mechanism. They establish relations between rules. In UPN formalism, a metarule is represented by a subnet. Metarules increase the efficiency of the rule control process. Grouping related rules together saves substantial amount of time in searching appropriate sets of rules which are due to be executed together and be invoked many times. In other words, a metarule reflects a restricted set of rules that are bound to be called upon together and repeatedly. Metarules are mainly used in UPN as a mechanism to define subnets which allow for the structural composition of the rule specification knowledge. They will be further discussed in the next section where our modeling approach is presented.
4.4 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. Following a similar 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 to the refinement of rules within each scenario, it is necessary to synthesize all scenarios for all entities to form a coherent net representing the company-wide policy in the system.

Some work in hierarchical representation using Petri nets has been done for various applications [Valette 79], [Suzuki 83], [Narahari 85], [Jeng 90]. The hierarchical modeling methodology facilitates the modeling task. It is discussed in detail in the following sections.

4.4.1 Top-down Stepwise Refinement Technique

The top-down stepwise refinement technique has been developed 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 is primitive transitions representing primitive rules, and the other is compound transitions representing 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. Two techniques for constructing compound transitions, representing metarules, are presented below.
Horizontal composition of rules: hmrules

Rules at the same level of abstraction can be connected to form subnets. This horizontal composition allows the aggregation of rules under specific criteria. Horizontal compositions are established by means of what we call "hmrules". A hmrule $h_{m_a}$, specifies a relation in a subnet with a set of transitions $\{t_1, t_2, \ldots, t_m\}$, where $m \geq 1$ and $t_i$ is defined at the level of abstraction $a$, $\forall t_i \in h_{m_a}$. A subnet, defined by metarule $h_{m_a}$, is composed by the set of transitions and the places that are interconnected together.

Hmrules are generally used to identify various scenarios used to refine a transition at a lower level of abstraction. This aggregation is very useful at the implementation stage, because a complete subnet is translated into one procedure, which avoids the use of local variables and makes the code more efficient. For example, if the complete subnet were implemented by several procedures, additional local variables would be required to pass the state of one procedure to the other, in order to complete the execution of the complete subnet without user interruption.

An example of a horizontal composition is one of the subnets for the work center creation scenario, related to the release of a work center in MRP II, shown in figure 4.1. The five rules related with that scenario are grouped together, represented by transition $t_2$ in figure 4.3 with its initial marking. The formal representation of that subnet is specified by its incidence functions as shown in tables 4.2 and 4.3. The color sets for the places are such that:

$$E = \{\varepsilon\}$$

$$MWCSC = WC\times DES \times DEP \times CAP \times MSTE \times RES \times ESD \times DES \times DEP \times CAP$$

$$WDDC = WC\times DES \times DEP \times CAP$$

Vertical composition of rules: vmrules
<table>
<thead>
<tr>
<th>$e$ for $I^*$</th>
<th>$t_{2.1}$</th>
<th>$t_{2.2}$</th>
<th>$t_{2.3}$</th>
<th>$t_{2.4}$</th>
<th>$t_{2.5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMwc WCID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMwc MWC</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPwc WCID</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPwc PWC</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.1}$ E</td>
<td>0</td>
<td>abs</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.2}$ WCID [wcid#]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.3}$ WCID [wcid#]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.4}$ WCID [wcid#]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.5}$ WDDC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Positive incidence functions.

<table>
<thead>
<tr>
<th>$I^-$</th>
<th>$t_{2.1}$</th>
<th>$t_{2.2}$</th>
<th>$t_{2.3}$</th>
<th>$t_{2.4}$</th>
<th>$t_{2.5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMwc WCID</td>
<td>0 [wcid#]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMwc MWC</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPwc WCID</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPwc PWC</td>
<td>0</td>
<td>abs</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.1}$ E</td>
<td>0 [wcid#]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.2}$ WCID [wcid#]</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.3}$ WCID [wcid#]</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.4}$ WCID [wcid#]</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{2.5}$ WDDC</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Negative incidence functions.
Figure 4.3: UPN graph of the scenario: "Creation of a work center via MRP II" at an abstract level with initial marking.
In order to facilitate the top-down stepwise refinement technique, UPN also allow for a vertical composition of rules. Vertical compositions in UPN are used as a mechanism to establish relations between one rule at a given level of abstraction and other rules which define knowledge at a lower level of abstraction. As a result, rules form an abstraction hierarchy. This abstraction facilitates the design of the model and it is in line with the natural way a company policy is defined: first with abstract and general rules, which are then refined with a higher degree of detail.

One rule can be refined and substituted by a set of rules representing more detailed specifications. This can be seen as a metarule where only part of its preconditions and postconditions are shown at the higher level of abstraction. Vertical composition is performed in such a way that rule preconditions and postconditions at a level of abstraction are preserved when working at the lower level. In UPN terminology, a compound transition is refined and replaced by a subnet, where incoming and outgoing arcs from the transition are maintained in the resulting subnet.

Given a compound transition, \( t_a \) (high level abstraction transition), and a level of abstraction \( i - 1 \), a vertical metarule (called "vmrule") \( vm_i^{t_a} \) is identified by the tuple \( < t_a, hmi > \), where \( hmi \) is a hmrule that identifies a subnet, \( UPN' \), which refines \( t_a \) at a lower level of abstraction \( i \).

The refinement of a compound transition of an abstract net produces a new UPN net which, in general, is the union of both nets minus the refined compound transition. A subnet being refined from a compound transition is formed with an attached calling protocol which establishes the link from the net at the abstract level with the subnet at the next lower level. The compound transition is replaced by the subnet with a calling net which contains one initiation transition (\( t_{init} \)), one waiting place (\( p_{wait} \)), and one returning transition (\( t_{ret} \)). In addition, \( p_{init} \) is the starting place of the subnet and \( p_{ret} \) is the ending place of it. One arc connecting
Figure 4.4: Example of a subnet and a calling protocol

\[ t_{\text{init}} \to p_{\text{init}} \text{ in the subnet and another arc connecting } p_{\text{ret}} \to t_{\text{ret}} \text{ in the subnet are added to link the calling protocol to the subnet. The calling protocol and the subnet are shown in figure 4.4.} \]

It is also possible to explode a \textit{compound} transition to several metarules. Let \( UPN_1 \) be a net representing specifications at a high abstract level and let \( UPN_{2_1}, \ldots, UPN_{2_n} \) be the subnets representing the metarules \( hm_{2_1}, \ldots, hm_{2_n} \), from \( < t_a, hm_{2_1}, \ldots, hm_{2_n} > \), that refine the \textit{compound} transition \( t_a \in T_1 \). The new \( UPN \) net is defined as follows:

1. \( P = P_1 \cup P_{2_1} \ldots \cup P_{2_n} \cup P_{\text{call}} \), where \( P_1, P_{2_1}, \ldots, P_{2_n} \) denote the sets of places of \( UPN_1, UPN_{2_1}, \ldots, UPN_{2_n} \) respectively, and \( P_{\text{call}} \) denotes the set of places of the UPN for the calling protocol.

2. \( T = \{T_1 \setminus \{t_a\}\} \cup T_{2_1} \ldots \cup T_{2_n} \cup T_{\text{call}} \), where \( T_1, T_{2_1}, \ldots, T_{2_n} \) denote the sets of transitions of \( UPN_1, UPN_{2_1}, \ldots, UPN_{2_n} \) respectively, and \( T_{\text{call}} \) denotes the set of transitions of the UPN for the calling protocol.
3. All the incoming and outgoing arcs are preserved, except the ones related with transition $t_a$:

$I^-(t, p) = I^-_{a}(t, p), \forall p \in P_1, \forall t \in \{T_1 - \{t_a\}\}$

$I^-(t, p) = I^-_{a_i}(t, p), \forall p \in P_i, \forall t \in T_i, 1 \leq i \leq n$

$I^-(t, p) = I^-_{call}(t, p), \forall p \in P_{call}, \forall t \in T_{call}$

Similarly for $I^+(t, p)$ and $I_a(t, p)$.

In addition,

$I^-(t_{init}, p) = I^-_{a}(t_{a}, p), \forall p \in P_1$

$I^+(t_{ret}, p) = I^+_{a}(t_{a}, p), \forall p \in P_1$

$I^+(t_{init}, p_{init}) = \bigcup\{I^-_{a_i}(t, p_{init}) \mid t \in T_{2i}, 1 \leq i \leq n\}$

$I^+(t_{ret}, p_{ret}) = \bigcup\{I^+_a(t, p_{ret}) \mid t \in T_{2i}, 1 \leq i \leq n\}$

$I^-(t, p) = \emptyset, \forall p \in P_{2i}, \forall t \in T_{2i}$, where $I^-_{a_i}(t, p) \cap I^-_{a}(t_a, p)$, and $I^-_{a_i}(t, p_{ret}) \neq \emptyset$

$I^+(t, p) = \emptyset, \forall p \in P_{2i}, \forall t \in T_{2i}$, where $I^+_a(t, p) \cap I^+_a(t_a, p)$, and $I^+_a(t, p_{ret}) \neq \emptyset$

4. $M(p) = M_1(p) + M_2(p), \forall p \in P_1 \cap P_{2i}, 1 \leq i \leq n$

$M(p) = M_1(p), \forall p \in \{P_1 - (P_1 \cap P_{2i}) - ... - (P_1 \cap P_{2n})\}$

$M(p) = M_2(p), \forall p \in \{P_{2i} - (P_1 \cap P_{2i})\}, 1 \leq i \leq n$

An example of the top-down refinement technique is the refinement of the rule "Release of a work center in MRP II (figure 4.1), which is the compound transition $t_2$ in the abstract scenario "Creation of a work center via MRP II (figure 4.3). During the refinement procedure a calling protocol is used, and the refined UPN is shown in figure 4.5 with initial marking. Preconditions and postconditions represented by transition $t_2$ are preserved (arcs to/from $EMwc, NPwc$ and $EPwc$).

4.4.2 Synthesis Technique

It is necessary to synthesize related scenarios to build the company wide policy, represented by one single net. There have been some synthesis techniques presented
Figure 4.5: Partially refined UPN of the scenario "Creation of a work center via MRP II" with initial marking.
in [Suzuki 83], [Narahari 85], [Jeng 90] based on their application domain. In our work we take advantage of the features of UPN, such as global places that reflect the state of the databases involved. In addition, we take advantage of standard modification procedures embedded in the database management systems associated with each application systems. The synthesis of UPN is achieved through the following mechanism:

**Global places (representing database states)**

Due to the representation of database states in UPN, every scenario involves checkings, updates and retrievals in some of the databases of the system. Therefore, connections from and to global places, which represent database states, exist in every UPN. These global places provide the connectivities between all scenarios. For example, the scenario "Creation of a work center via MRP II" (figure 4.3) and "Removal of a work center via MRP II" (figure 4.6) are synthesized through the work center global places of the databases in MRP II, CAPP, and SFC. The synthesized net is shown in figure 4.7.

Alternatively, standard modification procedures (representing the default procedures maintained in database management systems) can be used to synthesize subnets together. As shown in figure 4.8, there are three procedural levels in a typical database management system. The lowest level is the physical procedural level which represents the actual database changes. The procedures at this level are represented here by \textit{ins}, \textit{del}, and \textit{upd}. The second level is the modification procedural level which represents the modification procedures with build-in rules in the DBMS. The procedures at this level are represented here by \textit{insert}, \textit{delete}, and \textit{update}. The highest level is the application procedural level which represents the user defined procedures (\textit{create}, \textit{release}, \textit{remove}, etc) based on company policy and expert rules, that are the focal points of this work. Note that the modification procedural level could also be modeled using the same methodology.
Figure 4.6: UPN graph of the scenario: "Removal of a work center via MRP II at an abstract level with initial marking

and techniques presented in this chapter. In that case, instead of the global places defined in section 4.3.2, these modification procedures would become the links between application procedures which have been developed in various scenarios. The reasoning is that all application procedures call for the standard modification procedures embedded in the DBMS in order to make changes in the databases involved.

In order to avoid "re-inventing the wheel, we decided not to model the database management systems of the manufacturing application systems involved using UPN, but to define the database states as global variables and to interface the application procedures (company policy) through those global places representing database states.
Figure 4.7: UPN graph of the synthesized scenario: "Creation and removal of a work center via MRP II" at an abstract level with initial marking
Figure 4.8: Procedural levels in Database Management Systems

4.5 Conclusions

A formal structured representation schema for rule based systems has been developed and demonstrated to achieve information integration for manufacturing applications. The representation schema, called UPN, is based on the graphical and formal capabilities of colored Petri nets to express and validate if-then rules. The UPN is capable of representing user specification rules, as well as database updates and retrievals, which are necessary for controlling and integrating the information flow within current and future distributed database systems. Related rules can be aggregated at the same level of abstraction. The explosion of one rule at a given level of abstraction to a set of aggregated rules at a lower level of abstraction is also allowed. Finally, the synthesis of related models is achieved through the global database places.
Chapter 5

Knowledge Verification

5.1 Introduction

The major objective of creating a KBS using Petri nets is the ability of validating the KBS mathematically and systematically. The completeness (dead-end rules, unifiable rules), consistency (redundant rules, subsumed rules, under-constrained rules), and conflicts, are the major issues in knowledge base verification [Nguyen 87], [Lopez 90]. The incidence matrices of Petri nets representing the rule base can be used to perform some of these verification checks, which can be complemented by the user with the aid of specific domain knowledge. Several other analysis techniques for Petri nets, including reachability trees and net invariants, are also used [Peterson 81] [Jensen 86] [Martinez82]. 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 programs for computerizing these analysis methods have been developed and applied extensively as seen in chapter 7. Some reduction rules [Lee 85] have also been investigated to reduce the complexity of nets prior to the analysis phase.

However, these analysis techniques were initially developed for Generalized
Petri Nets (GPN), and do not apply to Colored Petri Nets (CPN) which are characterized by a great diversity of linear functions that are associated to their arcs. Therefore, unlike analysis algorithms for GPN that use integer matrices, analysis algorithms for CPN need to manipulate matrices composed by linear functions. This fact introduces high complexities in the development and execution of these algorithms. Due to this fact, a necessary step must be taken to convert the high level net of the UPN model into a low level net expressed in terms of Generalized Petri Nets (GPN).

According to the above discussion, the following two major aspects are considered:

**Structural verification** focuses on the correctness of the knowledge base structure, which mainly depends on the KBS representation formalism used. In our case, it is the structure of Petri nets. With a formal representation of the KBS, it is possible to verify the KBS structure mathematically. The structural verification does not depend on the domain which the KBS is applied to, or the rule specification language used for the implementation. Thus it is generic for all KBS using the same representation formalism. The following properties can be tested using structural methods. [Nguyen 87]

**Completeness** : The goal is to detect possible gaps in the rule base that have been overlooked in the modeling process.

**Consistency** : The goal is to perform a static analysis of the logical semantics of the rule structure.

**Domain knowledge verification** focuses on assuring a proper behavior with respect to the domain of the system model. It depends on the functionalities of the company policy and can be very different from one application to the other. Therefore, it can not be performed fully automatically; it needs user
Figure 5.1: Staged model verification.

input to determine the correctness of the functionalities of the system, and the compliance with the company policy.

Petri nets have provided traditionally the mathematical background to carry out this kind of verification. Generic analysis methods and properties of Petri nets are shown in the following section, which provide the basis of verifying the KBS.

The staged approach to our KBS verification system, shown in figure 5.1, consists of three major parts:

1. Generic analysis methods and properties of Petri nets

2. Structural Verification

3. Domain Knowledge Verification

The following sections describe first the generic analysis methods and properties of Petri nets, which can be used for verification. We then present the verification methodology, and finally some examples for demonstration purposes.
5.2 Conversion of UPN to Generalized Petri Nets

We call this conversion process: "abstraction and unfolding". The first step is to perform an abstraction on the identification number of a database entity, which if unfolded will create a number of duplicated nets, one for each possible value of its identification number. For example, unfolding the wcid of the work center record in MRP II will create a large number of nets, as many as the number of work centers which can be inserted into the database. In reality, we need only one net to represent the company policy for any work center. More specifically, we retain only one net to model the control of information flow in the system for one "generic" work center and we can use it for the verification of the KBS. The functionality of the model does not differ for different work centers. It is specified with respect to any work center in the system. Thus the potential variety of hundreds, if not more, work centers in a typical company can be reduced to one "generic" work center net, without harming the generality of the model. The second step in the net conversion process is to apply the formal Colored Petri net unfolding algorithm on a UPN, which consists of the information that is aggregated in the color sets of its places and transitions, and the functions of its arcs. The result is a new net with more places, transitions, and arcs, as described in section 5.2.3. The unfolding of the color sets in places and transitions is done using only the conditional data fields (eg. sts and ste of the work center record in MRP II). The rest of the data fields in the database record, which do not contribute to the control of the information flow in the system and are only used for carrying supplementary information, do not need to be unfolded (examples include work center description, work center capacity, etc.). The unfolded arcs of the net have only integer numbers associated with them (as opposed to linear functions) and tokens are colorless. The result of this conversion process is an analyzable GPN.
In order to describe the technical issues for abstracting and unfolding UPN, we first briefly introduce the GPN formalism and then propose a procedure to effect the conversion outlined above.

### 5.2.1 Petri Net Formalism

Formally, a Petri net can be defined as a 5-tuple $PN \equiv< P, T, I^+, I^-, M_0 >$ in which $P$ and $T$ are disjoint and non-empty sets (whose elements are named places and transitions respectively), $I^+$ and $I^-$ are mappings from $P \times T$ into $N$ called pre-incidence and post-incidence functions, and $M_0$ is a (initial) marking: $M_0 : P \rightarrow N$

For each $t \in T$, we define the set of input places and the set of output places as $I^-(t) = \{ p \in P \mid I^-(p, t) \neq 0 \}$, $I^+(t) = \{ p \in P \mid I^+(p, t) \neq 0 \}$, respectively.

We say that $t \in T$ is enabled when $\forall p \in P$ it occurs that $M(p) \geq I^-(p, t)$. In other words, if the number of tokens in each place of $I^-(t)$ is bigger or equal to the weight (i.e., an integer number) of the corresponding arc, then $t$ is enabled. An enabled transition can be fired, changing the net from the present marking $M$ to a new marking $M'$, given by $M'(p) = M(p) + I^+(p, t) - I^-(p, t), \forall p \in P$. The result of firing is the removal of tokens from each of the input places and the addition of tokens in each of the output places, and the number of tokens removed and added is equal to the weights of the corresponding arcs.

### Incidence Matrix

An incidence matrix ($I = I^+ - I^-$) is an algebraic representation of one Petri net with columns representing transitions, rows representing places, and the matrix elements representing the weights of arcs from transitions to places ($I^+$) and from places to transitions ($I^-$). It is the state space representation of the Petri net model, and it can be used to reconstruct the Petri net graph of the system modeled.
Assuming that all weights are equal to unity, the elements of the incidence matrix are either -1, 0, or 1, which reflects that the corresponding place is a precondition of, not connected to, or a postcondition of, the corresponding transition, respectively.

A state equation to transform one marking of a Petri net into the next one is shown below.

\[ M_i = M_{i-1} + Ix \]

where \( M_i \) and \( M_{i-1} \) are the markings after and before the firing of a set of transitions, \( x \) is a vector of a set of sequentially enabled transitions, and \( I \) is the incidence matrix.

### 5.2.2 Abstraction

In a manufacturing environment, the user provides generic specifications for work centers, parts, routings, without referencing any particular work center number, part number, or routing number respectively. In order to illustrate the abstraction process of an UPN, let us consider an example, the abstraction of one rule of the UPN presented in the previous chapter shown in figure 5.2(a). This rule is defined by transition \( t_1 \) and places \( p_1, NMwc \) and \( EMwc \). To make the abstraction on this net, the identification number of the work center record is removed from all the color sets of places and transitions and from the functions of the arcs. However, only the color sets of the conditional data (eg. \( wcid, sts \) and \( ste \) - see chapter 4 section 4.3.1) are considered, because these are the ones which are used to control the information flow in the net. The new net is shown in figure 5.2(b), where \( wcid \) has been removed from the color sets of places and transitions, and from the functions of arcs. The color set of place \( NMwc \) was changed from \( WCID \) to unity, and the color set of place \( EMwc \) was changed from \( WCID \times STS \times STE \) to \( STS \times STE \). Similarly, the color set of transition \( t_1 \) was changed from \( WCID \times \{sts = h\} \times \{ste = na\} \) to \( \{sts = h\} \times \{ste = na\} \). The
function of the arc between \( NMwc \) and \( t_1 \) was changed from \( Mwc(wcid = wcid\#) \) to unity, whereas the function of the arc between \( p_1 \) and \( t_1 \) need no change. Note that \( p_1 \) reflects the intention of a user to enter a work center record to the system, and as such it contains an uncolored token. Finally, the function of the arc between \( t_1 \) and \( EMwc \) was changed from \( Mwc(wcid = wcid\#, sts = h, ste = na) \) to \( Mwc(sts = h, ste = na) \). This new net preserves the behavior of the model for a "generic" work center. Therefore, a token flowing in the net represents a "generic" work center and reflects only all other color sets in the work center data structure except \( wcid \).

The incidence matrices of the net are changed from

\[
I^- = \begin{bmatrix}
NMwc \\
EMwc \\
p_1
\end{bmatrix} \begin{bmatrix}
t_1 \\
wcid = wcid\# \\
0 \\
1
\end{bmatrix},
I^+ = \begin{bmatrix}
NMwc \\
EMwc \\
p_1
\end{bmatrix} \begin{bmatrix}
t_1 \\
wcid = wcid\#, sts = h, ste = na \\
0 \\
0
\end{bmatrix}
\]

to

\[
I^- = \begin{bmatrix}
NMwc \\
EMwc \\
p_1
\end{bmatrix} \begin{bmatrix}
t_1 \\
1 \\
0 \\
1
\end{bmatrix},
I^+ = \begin{bmatrix}
NMwc \\
EMwc \\
p_1
\end{bmatrix} \begin{bmatrix}
t_1 \\
sts = h, ste = na \\
0 \\
0
\end{bmatrix}
\]

Note that \( I^- \) and \( I^+ \) still contain functions, instead of just 0 and 1.

### 5.2.3 Unfolding

The next step is to unfold the net to form a general Petri net which is analyzable. To construct the new unfolded net, we follow a step-wise process where places will be unfolded first, then transitions and finally arcs. We will show in detail how to unfold UPN to GPN by using the example of figure 5.2(b).

**Unfolding of Places**

Places \( NMwc \) and \( EMwc \) represent the nonexistence and the existence of a work center record in MRP II database, respectively. The unfolding of place \( NMwc \) is straightforward because its color set is already equal to unity. Therefore, \( NMwc \) will be unfolded to just one place representing the nonexistence of a
Figure 5.2: Conversion of a UPN rule to GPN.
"generic" work center. On the other hand, place \( EMwc \) contains two color sets of the work center record in MRP II, namely \( sts \) and \( ste \). In order to unfold the values of these fields in a net with uncolored tokens, we need to create one place for each possible combination of values in those fields which are defined in two separate data sets:

\[
MwcSTS = \{h, r\} \quad & \quad MwcSTE = \{na, av\}
\]

Following the formal unfolding technique for CPN, \( EMwc \) will be unfolded to a number of places equal to the Cartesian product of the color sets of those conditional fields (i.e., \( \{h, r\} \times \{na, av\} \)) as shown in figure 5.2(c). Each of those four different places represents one possible combination of values in the above data sets.

**Note:** Although three places (\( EMwc(sts = h, ste = av) \), \( EMwc(sts = r, ste = av) \), and \( EMwc(sts = r, ste = na) \)) are not connected to any transitions, they are kept temporarily, in case they are connected to other transitions in the subnet that contains \( t_1 \), otherwise they will be eliminated from these subnets.

**Unfolding of Transitions**

Each transition will be unfolded into a number of transitions equal to the number of possible color sets of that transition. In this case, the color set of transition \( t_1 \) contains only one color, \( sts = h, ste = na \), and therefore it is unfolded to one transition, as shown in figure 5.2(c).

**Unfolding of Arcs**

Each arc will be unfolded into a set of arcs with functions equal to unity and linked to places which are specified in the initial arc functions. In the above example, the arc between \( NMwc \) and \( t_1 \) has a function of unity and does not need to be unfolded; same for the arc between \( p_1 \) and \( t_1 \). The arc between \( t_1 \) and \( EMwc \) with an arc function \( Mwc(sts = h, ste = na) \) is unfolded into one arc connecting

68
the unfolded transition to the place of \( \{sts = h, ste = na\} \). The fact that this arc ends at that place and not any other one is determined by the initial function of this arc.

Following the unfolding algorithm, the incidence matrices of the net are changed from

\[
I^- = \begin{bmatrix}
NM_{wc} \\
EM_{wc} \\
p_1
\end{bmatrix} \begin{bmatrix}
t_1 \\
0 \\
1
\end{bmatrix}, \quad I^+ = \begin{bmatrix}
NM_{wc} \\
EM_{wc} \\
p_1
\end{bmatrix} \begin{bmatrix}
t_1 \\
sts = h, ste = na \\
0 \\
sts = h, ste = na \\
0
\end{bmatrix}
\]


to

\[
I^- = \begin{bmatrix}
NM_{wc} \\
EM_{wc}(sts = h, ste = na) \\
EM_{wc}(sts = h, ste = av) \\
EM_{wc}(sts = r, ste = na) \\
EM_{wc}(sts = r, ste = av)
\end{bmatrix} \begin{bmatrix}
t_1 \\
0 \\
0 \\
0 \\
1
\end{bmatrix}, \quad I^+ = \begin{bmatrix}
NM_{wc} \\
EM_{wc}(sts = h, ste = na) \\
EM_{wc}(sts = h, ste = av) \\
EM_{wc}(sts = r, ste = na) \\
EM_{wc}(sts = r, ste = av)
\end{bmatrix} \begin{bmatrix}
t_1 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]

Note now that all elements of the incidence matrices are equal to 0 or 1. The completely unfolded net, shown in figure 5.2(c), is now represented as a general Petri net and follows the firing rules of general Petri nets. For example, \( t_1 \) is enabled by at least one colorless tokens residing in each of its input places. Therefore, existing analysis methods for general Petri nets can be applied for the verification of our rule base.

5.3 General Net Analysis Methods and Properties

General net analysis methods, which are useful for the structural verification of General Petri nets (GPN) [Peterson 81] [Reisig 85] [Murata 89], are listed below.

- **Reachability tree**

  The reachability tree represents the reachability set of a Petri net. The root node represents the initial marking (or initial state). An arc, representing an enabled transition \( t_i \) under the initial marking, is drawn from the root
node to a new node, which represents a new marking after firing transition $t_i$. Each branch of the tree is then growing until no transition is fireable (terminal node).

The reachability tree can provide information about the net, such as deadlocks and unwanted final states. However, the complexity of the reachability tree is exponentially increasing with the size of the Petri net, with respect to the number of places, transitions, and arcs. This is still a potential research area in Petri net theory. Valuable experience of the complexity of generating reachability trees was gained from a project where we applied our methodology to model a set of chemical process change control procedures for MERCK CO. & INC [Garai 91]. That model contained 96 places and 60 transitions and it took more than 3 hours to generate one reachability tree for the net, on a Sun 3/60 workstation, as opposed to 30 seconds for a sample net with 20 places and 12 transitions.

- Invariants

The S-invariant represents a place vector of weighting factors, which if multiplied with a place vector of any possible marking in the reachability set, results in a constant and all the weighting factors are positive. If all the weighting factors are equal to 1 or 0 and the resulting constant equals to 1, then the places with positive weighting factors in the S-invariant are mutually exclusive. This set of places can never be marked simultaneously. This refers for example to the correct utilization of a shared resource, or the correct modeling of two mutually exclusive logical conditions. Solving the S-invariants can be done by multiplying both sides of equation (1), in section 5.2.1, by $y^T$, which is a place vector of unknown weighting factors.

$$y^T \cdot M_t = y^T \cdot M_{t-1} + y^T \cdot Ix$$

(2)
This is combined with the following equation indicating that the total number of tokens between two different markings is constant.

\[ y^T \cdot M_t = y^T \cdot M_{t-1} \]  \hspace{1cm} (3)

The S-invariants can then be derived from the following equation by solving for \( y^T \):

\[ y^T \cdot I = 0 \]  \hspace{1cm} (4)

and \( y^T \geq 0 \).

The solution is usually infinite and a fast algorithm in solving the minimal set of S-invariants (i.e., linearly independent) was proposed by [Martinez82].

Therefore, by utilizing the existing analysis methods of Petri net theory and some basic linear algebra, we can verify some properties of the knowledge base. The following sections describe some other types of problems which can be detected in Knowledge Based Systems.

### 5.4 Checking Completeness

Completeness verification involves the revelation of unreachable places or unfirable transitions in the rule base.

- **Unfirable transitions**

  A transition, which will never fire from any possible state of the system, is an unfirable transition. It happens usually because of over-constrained preconditions. This can be detected by checking the reachability tree of the Petri net representing the KBS with a finite set of possible initial markings determined by the model designer.

- **Unreachable places**
A place, which will never be marked throughout the life cycle of the system, is an unreachable place. This is due to a modeling error, unless that place corresponds to an input place, i.e., an interface of the system with its surroundings. An unreachable place is one which does not have any input arcs and is not an input place (see chapter 4 section 4.3.2). Usually, unreachable places are those whose input transitions are all unifiable. This can be detected from the incidence matrices of the Petri net representing the KBS as follows:

For $PN \equiv (P, T, I^+, I^-, M_0)$, if $I^+(p_i, t) = 0, \forall t \in T$ and $p_i \notin P_{\text{input}}$, $p_i$ is then an unreachable place, where $P_{\text{input}}$ is the set of input places of $PN$.

- **Dead-end places**

A place, which will never be unmarked once it is marked, is a dead-end place. This is due to a modeling error, unless that place corresponds to an output place, meaning an interface with the user. A dead-end place is then a place which does not have output arcs and is not an output place. This can be detected from the incidence matrix of the Petri net representing the KBS. In addition, places whose output transitions are all unifiable, are also considered as dead-end places.

For $PN \equiv (P, T, I^+, I^-, M_0)$, if $I^-(p_i, t) = 0, \forall t \in T$ and $p_i \notin P_{\text{output}}$, $p_i$ is then a dead-end place, where $P_{\text{output}}$ is the set of output places of $PN$.

### 5.5 Checking Consistency

One of the most critical issues in the design of a knowledge base is the consistency check, which eliminates conflicts and redundancies in it. Some of the properties related to the consistency of rules are listed and discussed below.

Some simple cases of such problems can be detected by using incidence matrices
of Petri nets representing KBS's, based on an idea initially suggested by the CHECK system. It is used for verification by matching facts and rules in KBS's [Nguyen 87]. The advantage of using Petri nets, as opposed to dealing directly with the facts and rules in a knowledge base, is that the relations between facts and rules are now represented explicitly in the structure of a Petri net. This eliminates the effort for searching and matching facts and rules in some kind of a rule specification language.

![Diagram of Petri net](image)

Figure 5.3: Simple redundant rules.

- **Redundant rules**

Rules are redundant if they succeed from the same preconditions and have the same results. e.g.:

\[
\begin{align*}
&\text{If } a \text{ then } b \\
&\text{If } a \text{ then } b
\end{align*}
\]

An example of a simple redundant rule in a Petri net form is shown in figure 5.3.

If two transitions have the same set of input places and the same set of output places, they are then redundant. Redundant rules can be found from the incidence matrices where transitions \( t_x, t_y \) have identical input place and output place sets: \( i = 0, \forall i \in \{I^- (t_x) - I^- (t_y)\} \) and \( \forall i \in \{I^+ (t_x) - I^+ (t_y)\} \).

The incidence matrices shown below represent the net in figure 5.3.

\[
I^- = \begin{bmatrix} a \quad t_1 & t_2 \\
\text{ } b & 1 & 1 \\
\end{bmatrix}, \quad I^+ = \begin{bmatrix} a \quad t_1 \\
\text{ } b & 0 & 0 \\
\end{bmatrix}
\]

73
By computing $I^-(t_1) - I^-(t_2) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $I^+(t_1) - I^+(t_2) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, we conclude that $t_1$ and $t_2$ are redundant.

![Figure 5.4: Subsumed Rules.](image)

- Subsumed rules

Two rules are subsumed if they have the same results but one contains additional constraints than the other. e.g.:

$$\begin{cases} \text{If } a \text{ then } c \\ \text{If } a \text{ and } b \text{ then } c \end{cases}$$

This happens if two transitions have the same output places, and the input places of one transition is a subset of the input places of the other one. This can be represented by the Petri net graph in figure 5.4.

Subsumed rules can be detected from the incidence matrices where transitions $t_x, t_y$ have identical output place sets and the input place set of $t_y$ is a sub set of the input place set of $t_x$: $i \geq 0, \forall i \in \{I^-(t_x) - I^-(t_y)\}$ and $\sum i \neq 0$, and $i = 0, \forall i \in \{I^+(t_x) - I^+(t_y)\}$. The incidence matrices shown below represent the net in figure 5.4.

$$I^- = \begin{bmatrix} a & t_1 & t_2 \\ b & 1 & 1 \\ c & 0 & 0 \end{bmatrix}, \quad I^+ = \begin{bmatrix} a & t_1 & t_2 \\ b & 0 & 0 \\ c & 1 & 1 \end{bmatrix}$$

By computing $I^-(t_1) - I^-(t_2) = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$ and $I^+(t_1) - I^+(t_2) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$, we conclude that $t_2$ is subsumed to $t_1$. 

74
5.6 Domain Knowledge Verification

Assuming the correctness of the rule based system structure, the system domain verification amounts to assure that the model of the domain embodied in the rule base complies with the semantics of the real world [Lopez 90].

- Data Constraints

Each of the data records in the data base, such as work center records, part records, and routing records, cannot be at different statuses at the same time; potential statuses of a work center record are mutually exclusive. For example, a work center cannot be at hold and released statuses at the same time.

These mutually exclusive properties can be partially detected by the place invariants (s-invariants), as described in section 5.3, if the total number of the tokens in the invariant is 1. From this set of s-invariants, mutually exclusive conditions can be checked against the logical content of the places of the model. For example, a net involving the work center record should have an s-invariant which consists of all the possible statuses of a work center.

- Free choice rules

These are rules succeeding from the same preconditions but their results are not contradictory. e.g.
This can be represented by the Petri net graph in figure 5.5.

If two transitions, with the same input places and before more constraints are introduced, result in non-contradictory output places, these transitions are choice free by nature. This is not allowed within our current domain knowledge, which is restricted to single paths for a specific marking of the net. This is called a \textit{decision free} net. The decision making is made within the application systems (CAD, CAPP, MRP II, and SFC).

- **Conflicting rules**

  Rules are conflicting if they succeed from the same preconditions but have conflicting results, based on specific domain knowledge. e.g.:

  \[
  \begin{align*}
  & If \ a \ then \ b \\
  & If \ a \ then \ c \\
  & b \ and \ c \ are \ conflicting
  \end{align*}
  \]

  This can be represented by the Petri net graph in figure 5.6.

  If two transitions, with the same input places, and before more constraints are introduced, result in a contradictory output, these transitions are conflicting by nature. Some of these conflicting rules can be detected by using the
incidence matrices and comparing the output place sets with the user specifications. Therefore, we look for transitions $t_x, t_y$ that have identical input place sets and unequal output place sets: $i = 0, \forall i \in \{I^-(t_x) - I^-(t_y)\}$, and $\exists i \neq 0, \forall i \in \{I^+(t_x) - I^+(t_y)\}$. The incidence matrices shown below represent the net in figure 5.6.

$$I^- = \begin{bmatrix} a & t_1 & t_2 \\ b & 1 & 1 \\ c & 0 & 0 \end{bmatrix}, \quad I^+ = \begin{bmatrix} a & t_1 & t_2 \\ b & 0 & 0 \\ c & 1 & 0 \end{bmatrix}$$

By computing $I^-(t_1) - I^-(t_2) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ and $I^+(t_1) - I^+(t_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, we conclude that $t_1$ and $t_2$ might be conflicting with each other; the user should look at the logical content of the output place sets.

### 5.7 Structural Verification of Complex Rules

The knowledge base verification cases discussed above, which have also been studied in various research projects [Nguyen 87] [Lopez 90] can only handle the verification of simple rules but not complex cases as the examples described below.

A more complicated case of redundant rules involves one primitive transition and one or more compound transitions. For example, let us consider two sub-nets $N_1$ and $N_2$ that have the same set of input places which are not output places of any transition in those nets, and the same set of output places which are not input places of any transition in those nets. These two nets are then redundant. See figure 5.3 where $t1$ becomes $N_1$ and $t2$ becomes $N_2$.

Our experience in modeling a knowledge base using Petri nets, indicates that it is possible to verify some complicated rules by applying restricted reduction rules. There has been some work done in Petri net reduction methods [Lee 85].

The procedure to apply these reduction rules on the Petri nets which represent
our Knowledge Base System is described below with an example shown in figure 5.7:

![Diagram](image)

Figure 5.7: Complicated redundant rules.

1. Find a place which has only one output transition: place $b$.

2. The output transition of this place should not have any other input place:

   $$I^-(a, t_3) = 0, I^-(b, t_3) = 1, I^-(c, t_3) = 0.$$  

3. Then this place can be eliminated and its output transition can be merged into its input transition $(t_{2,3})$. The output places $(c)$ of the merged transition will then be connected to the merged transition $(t_{2,3})$. The reduced net is shown in figure 5.8, which can then be checked for the redundancy of rules $t_1$ and $t_{2,3}$.

### 5.8 Example

![Diagram](image)

Figure 5.8: Reduced net.
Figure 5.9: Unfolded UPN of the subnet: "Release of a work center in MRP II".

The model verification procedures described in the previous sections are applied in the example shown in figure 4.1. The abstracted and unfolded net of figure 4.1 is shown in figure 5.9 and attention should be paid to the way the database places have been unfolded; any places that do not connect to any transitions within the net have been eliminated. For example, the database place \( EMwc \) was abstracted and unfolded into four places: \( EMwc(sts = h, ste = na) \), \( EMwc(sts = h, ste = av) \), \( EMwc(sts = r, ste = na) \), and \( EMwc(sts = r, ste = av) \). Of those, only three places \( EMwc(sts = h, ste = na) \), \( EMwc(sts = r, ste = na) \), \( EMwc(sts = r, ste = av) \) have been retained and used in the net. Similarly, the database place \( EPwc \) was abstracted and unfolded into three places: \( EPwc(sts = w) \), \( EPwc(sts = h) \), and \( EPwc(sts = r) \), and only one place \( EPwc(sts = w) \) has been retained and used in the net.

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. For the above example, there are 10 invariants found as shown below and they are all correct after being checked against the company policy. For example, invariant number 7 represents that a work center can not be at hold and released statuses at the same time.

1. $NM_{wc}$
2. $EM_{wc}(sts = r, ste = av)$
3. $NP_{wc, pret}$
4. $EM_{wc}(sts = h, ste = na), pret$
5. $p_{init}, p_{2,2}, p_{2,4}, p_{2,5}, pret$
6. $NP_{wc}, EM_{wc}(sts = r, ste = na)$
7. $EM_{wc}(sts = h, ste = na), EM_{wc}(sts = r, ste = na)$
8. $p_{init}, p_{2,2}, p_{2,4}, p_{2,5}, EM_{wc}(sts = r, ste = na)$
9. $NP_{wc}, EP_{wc}(sts = w)$
10. $p_{init}, p_{2,2}, p_{2,4}, p_{2,5}, EP_{wc}(sts = w)$

One reachability tree has been generated for the above net with the following initial marking: one token in each of the places $p_{init}, EM_{wc}(sts = h, ste = na)$, and $NP_{wc}$ as shown in figure 5.10. There should be only one final marking in the tree: one token in each of the following places, $EM_{wc}(sts = r, ste = na), EP_{wc}(sts = w), pret$. The reachability tree is shown in figure 5.11, which displays the number of tokens in each of the places ($p_{init}, p_{2,2}, p_{2,3}, p_{2,4}, p_{2,5}, NM_{wc}, EM_{wc}(sts = h, ste = na), EM_{wc}(sts = r, ste = na), EM_{wc}(sts = r, ste = av); NP_{wc}, and EP_{wc}(sts = w)$). It is used to detect deadlocks in the model. There is no deadlock detected in this case since the desired final marking was reached. On the other hand, the verification process reveals no redundant, free-choice, subsumed, or conflicting rules.
Figure 5.10: Marked PN of the subnet: "Release of a work center in MRP II".

\[
\begin{align*}
(1,0,0,0,0,0,0,1,0,0,1,0) \\
\downarrow t_{2,1} \\
(0,1,0,0,0,0,1,0,0,1,0) \\
\downarrow t_{2,4} \\
(0,0,0,1,0,0,1,0,0,1,0) \\
\downarrow t_{2,5} \\
(0,0,0,0,1,0,0,1,0,0,1)
\end{align*}
\]

Figure 5.11: Reachability tree generated for the net in figure 5.10.
5.9 Conclusion

The procedure of knowledge base verification involves iterations of applying the analysis methods on a Petri net model, feeding back results to the user, and resolving inconsistencies and incompleteness in the knowledge base before a satisfactory result is reached. A procedure to convert UPN into GPN for verification purposes was developed and procedures to manipulate incidence matrices to reveal potential redundant, free-choice, subsumed, and conflict rules have also been presented in this chapter. Whenever a new rule set is introduced into the knowledge base, it should be checked against the existing rules, through the procedures described above.
Chapter 6

Translation of UPN to Update Dependencies Language (UDL)

6.1 Introduction

The emphasis in this chapter is placed 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 purpose 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, [Biemans 86], 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 [Su 86]. 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
[Hardwick 89] [Spoon 90] and KRON (Knowledge Representation Oriented Nets)
by the University of Zaragoza [Muro 89] are some examples. The Update
Dependencies Language (UDL) [Mark 87] 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 chapter is structured as follows. To provide the reader with a better
understanding of the features of the UDL, we chose to outline its basic syntax and
semantics in the second section. The third section presents the correspondence
between UPN and UDL. The fourth section details the implementation strategy
and the translation procedure. The fifth section provides examples of the automatic
translation between UPN models 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.

6.2 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. The rest of this chapter, in addition to the translation algorithm from UPN to UDL, provides a number of examples of how the scenario used throughout this dissertation is translated into the formalism presented here.

### 6.2.1 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 work, automatically generated by the translation from UPN to UDL.

*Procedures* have the following form:

\[
OR(A_1 = V_1, \ldots, A_n = V_n; A_1 = W_1, \ldots, A_n = W_n) \\
\rightarrow C_1, O_{1,1}, \ldots, O_{1,n_1}. \\
\rightarrow \ldots \\
\rightarrow C_m, O_{m,1}, \ldots, O_{m,n_m}.
\]

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: \textit{insert} and \textit{update}. The example of releasing a work center record in MRP II, is shown in figure 6.3 and discussed in detail below.

The \textit{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 \( O_{i,1}, \ldots O_{i,n_i}, 1 \leq i \leq m \).

\textit{Conditions} are safe expressions formed through conjunction and negation of the following atoms (parenthesis are used to alter the default precedence of operators):

- \textit{Tuple existence tests} with the form, \( R(A_1 = V_1, \ldots, A_k = V_k) \), 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 represented in the following form: \( \sim R(A_1 = V_1, \ldots, A_k = V_k) \). 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 $\theta$ is a comparison operator ($<$, $\leq$, $=$, $\geq$, $>$) and $X$ and $Y$ are constants or variables. A comparison evaluates to true if the algebraic relation $\theta$ holds between $X$ and $Y$.

• The *empty* condition. It always evaluates to true.

• **Negative or positive variable instantiation tests** with the form, $\text{var}(V_i)$ or $\text{nonvar}(V_i)$, 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, $\text{var}(\text{Wcid})$ and $\text{nonvar}(\text{Wcid})$ are used to test the negative and positive instantiation of the variable Wcid.

• **Existential qualification**, $\exists V_1...V_n C$. 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)

  \[
  \langle \text{user defined name} \rangle R(A_1 = e_1, ..., A_k = e_k[; A_1 = f_1, ..., A_k = f_k])
  \]

  In the above example, the application procedure involved is:

  \text{release Wcd(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:

\text{insert Pwc}(\text{wcid=Wcid, des=Des, dep=Dep, cap=Cap, sts=w})

- \text{update procedure invocations} have the form:

\text{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:

\text{update Mwc}(\text{wcid=Wcid, sts=h; wcid=Wcid, cap=Cap, sts=r})

- \text{physical insertion, deletion, and update invocations} have the forms:

  \text{ins } R(A_1 = e_1, ..., A_n = e_n),

  \text{del } R(A_1 = e_1, ..., A_n = e_n), \text{ and}

  \text{upd } R(A_1 = e_1, ..., A_n = e_n; A_1 = f_1, ..., A_n = f_n).

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

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

\text{write('Enter wcid')}

\text{read(Wcid)}

The procedure abstraction/encapsulation hierarchy enforced by the syntax of the update dependency formalism is illustrated in figure 6.1. 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.
Figure 6.1: Procedure abstraction/encapsulation hierarchy.
Notice that physical insertion, deletion, and update, \textit{ins}, \textit{del}, and \textit{upd}, respectively, on a relation \textit{R} can only be invoked from within insertion, deletion, and update procedures on the relation \textit{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 the examples presented in section 6.5, 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 assumed that procedures \textit{insert}, \textit{delete}, and \textit{update} act as operations \textit{ins}, \textit{del}, and \textit{upd}, respectively.

### 6.2.2 UDL Semantics

The execution of a procedure can be depicted by an AND/OR graph (figure 6.2). The \textit{AND nodes} are those whose executions are tied together by an arc; the \textit{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 alternatives 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.

6.3 Feature Correspondence Between UPN and UDL

This section describes the translation of particular features of UPN to UDL.

6.3.1 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:

$$R(A_1 = V_1, \ldots, A_k = V_k).$$

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 $\text{Mwc(wcid, des, dep, cap, sts, ste, res, esd)}$):

$$\text{Mwc(wcid=lt101, des=lathe, dep=machining, cap=null, sts=h, ste=na, res=M12, esd=null)}$$
6.3.2 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: The MRP II user starts releasing a work center with $wcid = lt101$ following the rules specified in figure 4.1. 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: $\text{Mwc}(wcid=lt101,sts=h)$

On the other hand, the non-existence of the work center $lt101$ corresponds to the UPN place $\text{NMwc}$ of the MRP II database, with the function $wcid = lt101$; this can be translated into the UDL form of: $\sim \text{Mwc}(wcid=lt101)$

**Reasoning process status**: Generally corresponds to the states of an UDL application procedure. For example, places $p_{init}, p_2, p_3, p_4, p_5, p_{ret}$ in figure 4.1.

6.3.3 Database Related Arc Conditions in UPN as UDL Checking 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 4.1 has two arcs to and from place \( EMwc \) (in the MRP II database) with the same arc expression: \( wcid = wcid\#, \text{sts} = r \). This is translated into UDL form as:

\[
\text{Mw}(\text{wcid}=\text{wcid}, \text{sts}=r)
\]

- **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 \text{insert}(< \text{relation name }>(< \text{tuple spec }>))\. For example, transition \( t_5 \) in figure 4.1 has one arc from place \( NPwc \) and one to place \( EPwc \) (in the CAPP database) with the arc expression \( Pwc(wcid = wcid\#, \text{des} = \text{des}\#, \text{dep} = \text{dep}\#, \text{cap} = \text{cap}\#, \text{sts} = w) \). This is translated into UDL form as:

\[
\text{insert } Pwc(\text{wcid}=\text{wcid, des}=\text{des, dep}=\text{dep, cap}=\text{cap, sts}=w)
\]

- **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 4.1 implies an update to the record \( M_{wc} \) (in place \( EM_{wc} \)) in the MRP II database that is translated into UDL form as:

\[
\text{update}(M_{wc}(\text{wcid}=\text{wcid}; \text{wcid}=\text{wcid}, \text{cap}=\text{Cap}, \text{sts}=\text{r}))
\]

### 6.3.4 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 4.1 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:

\[
\text{read (Wcid)}
\]
For better legibility, a message like the following can be printed to prompt the user:

```plaintext
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 4.1 is translated as an error message for the work center identification provided in variable `wcid#`:

```plaintext
write('Output in P4 for data: ' wcid#)
```

or, if the place has an associated label:

```plaintext
write('work center already exists: ' wcid#)
```

### 6.3.5 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 execute a series of transition firings without user interruptions. For example, if $n$ number of transitions are designed to be fired sequentially, without any user input and they are implemented into individual procedures, at least $n - 1$ new variables...
representing the completion of the first \( n - 1 \) transition firings have to be created. Secondly, this 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.

An actual example is shown below: in order to implement transition \( t_3 \) from figure 4.1 as a composed operation, we need a new variable, \texttt{varP2}, to test the value \texttt{Wcid} in place \( P2 \rightarrow \)

\[
\text{Release-Transition-}t_3 \quad \text{Mwc(wcid=Wcid, sts=Sts)} \\
\rightarrow (\text{Wcid = varP2}) \land \text{Mwc(wcid=Wcid, sts=r)}, \quad \text{write('Work center already has ''r'' status in MRP II').}
\]

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, as mentioned above. Therefore, we decided, as a general rule, 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 involved) to implement the transition status of the net execution.

To take advantages of these features, we follow what we call an information driven approach. According to the UDL syntax, only one procedure alternative (transition) of a procedure call can be successfully executed. This presents a limitation for the UPN execution syntax, according to which one transition can be enabled and fired automatically following the firing of its preceding transition. To eliminate this limitation, we use a recursive call to the same procedure, with the necessary parameters carried over. An example is shown below which has three procedure alternatives. The first as well as the second alternative of this procedure
will invoke a recursive call, which execute another copy of the same procedure with initiated parameter values, after all its preconditions are evaluated to be true and all its operations are successfully executed. This enables the continuous execution of more than one alternative within one UDL procedure. The need for recursive calls in UPN models is identified if the outgoing arcs of a transition go to places internal to the subnet that do not belong to database places (i.e., places of local scope - chapter 4, section 4.3.2). This means that other transitions within the same subnet may continue firing uninterrupted. It is obvious when the recursive call sequence has finished. That happens when the procedure call reaches an alternative which do not have recursive call. In this example, the third alternative will end the recursive call sequence.

\[ OR(A_1 = V_1, \ldots, A_n = V_n) \]
\[ \rightarrow C_1. \]
\[ O_{1,1}, \]
\[ O_{1,2}, \]
\[ OR(A_1 = V_1, \ldots, A_n = V_n), \]
\[ \rightarrow C_2. \]
\[ O_{2,1}, \]
\[ OR(A_1 = V_1, \ldots, A_n = V_n), \]
\[ \rightarrow C_3. \]
\[ O_{3,1}, \]
\[ O_{3,2}, \]

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

98
executed.

6.4 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, and 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 appears 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 6.3.3. The conjunction of these checking 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 checking recognized above \((Var_i, ..., Var_j)\), and generate a positive variable instantiation test for each one. The conjunction of these tests is another precondition:

\[ \text{nonvar}(V_i) \land \ldots \land \text{nonvar}(V_j) \]

(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:

\[ \text{var}(V_x) \land \ldots \land \text{var}(V_y) \]

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 6.3.4. For each variable that needs to be provided from the user, generate the appropriate input sequence \((<\text{Text } V_p>)\) that corresponds to the interpretation of the attribute name bound by \(V_p\) in the database record tables:

\[ \text{write('Enter < Text } V_p >'), \text{read}(V_p), \]

100
For each output variable generate: \texttt{write(' < place label text > ')}

(b) Recognize deletion, insertion and update UDL modification procedures, as explained in section 6.3.3 and generate the appropriate invocations:

\texttt{delete(< relation name >(< tuple spec >))}
\texttt{insert(< relation name >(< tuple spec >))}
\texttt{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 done by reading the content (name) associated to the transition, in the case of a \textit{compound} transition.

(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 other transition 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.

\texttt{< O > < R > (A_i = V_i, ..., A_j = V_j)}

6.5 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 6.4, 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.

6.5.1 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 4.1, which was used to illustrate the creation of UPN models described in chapter 4, section 4.2. 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 4.1):

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:
   $< O > \leftarrow \text{release} \ (\text{metarule name})$
   $< R > \leftarrow \text{Mwc} \ (\text{corresponding database record})$

Attribute names that appear in the arc expressions are:
wcid, des, dep, cap, sts
and their corresponding variables \((\text{wcid\#}, \text{des\#}, \text{dep\#}, \text{cap\#})\) are modified into the following UDL variable syntax: \(\text{Wcid}, \text{Des}, \text{Dep}, \text{Cap}\).

The procedure heading becomes \(\Rightarrow\)

\(\text{release } Mwc (\text{wcid=Wcid, des=Des, dep=Dep, cap=Cap})\)

2. Conditions for the alternatives:

\(t_1\) There is no connection with database places (rows \(N M\ wc\) and \(E M\ wc\) 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 (\(\text{Wcid}, \text{Des}, \text{Dep}\) and \(\text{Sts}\)) 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 an arc expression \(\text{wcid\#}\). This means that in the incoming arcs to this transition a work center identification number (variable \(\text{Wcid}\)) was not provided, but will be provided to the outgoing arc. In order to reduce the code, only this test is really needed: \(\text{var(Wcid)}\). The complete condition is \(\Rightarrow\)

\(\text{var(Wcid)}\)

\(t_2\) It has incoming and outgoing arcs to \(N M\ wc\) (MRP II database) with the same arc expression \(Mwc(wcid = \text{wcid\#})\). This is a checking for the non existence of \(Mwc\) with that specific work center identification number \(\Rightarrow \sim Mwc(\text{wcid=Wcid})\). It has another incoming arc with \(\text{wcid\#}\) from \(p_2\) providing the work center id. This information must be checked for positive instantiation \(\Rightarrow \text{nonvar(Wcid)}\). 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

\[ \Rightarrow \]
\[ \text{nonvar(WcId)} \land \neg M_{\text{wc}}(\text{wcid}=\text{wcid}) \]

\( t_3 \) Similarly, the complete condition is \( \Rightarrow \)
\[ \text{nonvar(WcId)} \land M_{\text{wc}}(\text{wcid}=\text{wcid}, \text{sts}=r) \]

\( t_4 \) Similarly, the complete condition is \( \Rightarrow \)
\[ \text{nonvar(WcId)} \land \text{var(Cap)} \land \]
\[ M_{\text{wc}}(\text{wcid}=\text{wcid}, \text{des}=\text{Des}, \text{dep}=\text{Dep}, \text{sts}=h, \text{ste}=\text{na}) \]

\( t_5 \) Similarly, the complete condition is \( \Rightarrow \)
\[ \text{nonvar(WcId)} \land \text{nonvar(Des)} \land \text{nonvar(Dep)} \land \text{nonvar(Cap)} \land \]
\[ M_{\text{wc}}(\text{wcid}=\text{wcid}, \text{sts}=h) \]

3. Operations for the alternatives:

\( t_1 \) Column \( t_1 \) from \( I^- \) and \( I^+ \) shows that variable \( \text{wcid}\# \) needs to be requested (there are no incoming variables and variable \( \text{wcid}\# \) is outgoing) \( \Rightarrow \)
\[ \text{write('Enter wcid'),} \]
\[ \text{read(WcId),} \]
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 \( \text{WcId} \) \( \Rightarrow \))
\[ \text{release M}_{\text{wc}}(\text{wcid}=\text{wcid}) \]

\( t_2 \) 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,
\textit{wcid}\#, and the text associated with the interpretation of \textit{p}\textsubscript{3} must be displayed \Rightarrow

\texttt{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 \textit{p}\textsubscript{1} has no variables) \Rightarrow

\texttt{release Mwc()}

\textbf{t}\textsubscript{3} Only an output statement is needed to display the information in the arc expression, \textit{wcid}\#, and the text associated to the interpretation of \textit{p}\textsubscript{4} \Rightarrow

\texttt{write('Work center already has 'r' status in MRP II', Wcid),}

No new call is needed because the output place \textit{p}\textsubscript{4} is not connected to any other transition.

\textbf{t}\textsubscript{4} The input for variable \textit{Cap} is required and then a recursive call is made with the information for the \textit{wcid}, \textit{des}, \textit{dep}, \textit{cap} and \textit{sts} parameters \Rightarrow

\texttt{write('Enter capacity'),}

\texttt{read(Cap),}

\texttt{release Mwc (wcid=Wcid, des=Des, dep=Dep, cap=Cap).}

\textbf{t}\textsubscript{5} An update modification procedure can be identified because there is an arc coming from the database place \textit{EM}\textit{wc} with a different function \((Mwc(wcid = wcid\#))\) to the one that is going back to \textit{EM}\textit{wc} \((Mwc(wcid = wcid\#, des = des\#, dep = dep\#, cap = cap\#, sts = r))\) \Rightarrow

\texttt{update Mwc(wcid=Wcid, sts=0; wcid=Wcid, cap=Cap, sts=r),}

It also has an associated procedure call \Rightarrow

\texttt{insert Pwc(wcid=Wcid, des=Des, dep=Dep, cap=Cap, sts=w).}
release Wwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap)

→ var(Wcid),
    write('Enter wcid'),
    read(Wcid),
    release Wwc(wcid=Wcid).

→ nonvar(Wcid) ∧ ¬Wwc(wcid=Wcid),
    write('Work center ID does not exist in MRP II, enter again', Wcid),
    release Wwc().

→ nonvar(Wcid) ∧ Wwc(wcid=Wcid,sts=r),
    write('Work center already has 'r' status in MRP II', Wcid),

→ nonvar(Wcid) ∧ var(Cap) ∧ Wwc(wcid=Wcid,des=Des,dep=Dep,sts=h,ste=na),
    write('Enter capacity'),
    read(Cap),
    release Wwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap).

→ nonvar(Wcid) ∧ nonvar(Des) ∧ nonvar(Dep) ∧ nonvar(Cap),
    update Wwc(wcid=Wcid,sts=h;wcid=Wcid,cap=Cap,sts=r),
    insert Pwc(wcid=Wcid,des=Des,dep=Dep,cap=Cap,sts=w).

Figure 6.3: UDL code for the scenario “Release of a work center in MRP II”.

The final UDL code resulting from this translation is shown in figure 6.3.

6.5.2 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 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 6.4 and then further refined down to a more detailed level, as shown in figure 6.5. 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 6.5). The top-down refinement technique used was discussed in chapter 4, section 4.4.

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
Figure 6.5: Subnet of the scenario “Deletion of a work center in MRP II".
UDL application procedures are generated as shown below.

(1). UPN subnet no. 1

During the translation of operations $t_2$, the following two application procedures called by it have to be satisfied, before any other modification procedures can be implemented $\Rightarrow$

\[ \text{check.1 } M_{wc} (wcid=\text{Wcid}), \text{ check.2 } P_{rout} (wcid=\text{Wcid}, psts=Psts). \]

Three deletion modification procedures can be identified: an arc coming from the database place $EM_{wc}$ with the expression $M_{wc}(wcid = wcid\#)$ and another one going to the database place $NM_{wc}$ with the same expression (same is the case for CAPP and SFC) $\Rightarrow$

\[ \text{delete } M_{wc}(wcid=\text{Wcid}), \text{ delete } P_{wc}(wcid=\text{Wcid}), \text{ delete } S_{wc}(wcid=\text{Wcid}). \]

(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 checking, which are represented by the inhibitor arcs, for the non-existence of any routing $E_{rout} (wcid=\text{Wcid}, psts=h)$ using that specific work center identification number and bearing an h or r status $\Rightarrow$

$\sim E_{rout} (wcid=\text{Wcid}, psts=h)$ and $\sim E_{rout} (wcid=\text{Wcid}, psts=r)$.

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

\[ \text{nonvar(Wcid)} \land \sim E_{rout} (wcid=\text{Wcid}, psts=h) \land \sim E_{rout} (wcid=\text{Wcid}, psts=r), \]

The full UDL code for this multi-procedural subnet is presented in figure 6.6.
remove Mwc(wcid=Wcid)

\[\rightarrow \text{var}(Wcid),\]
\[\text{write}(\text{'Enter wcid'}),\]
\[\text{read}(Wcid),\]
\[\text{remove Mwc}(wcid=Wcid).\]

\[\rightarrow \text{nonvar}(Wcid),\]
\[\text{check1.rmv.mwc Mwc (wcid=Wcid),}\]
\[\text{check2.rmv.mwc Prout (wcid=Wcid,psts=Psts),}\]
\[\text{delete Mwc}(wcid=Wcid),\]
\[\text{delete Pwc}(wcid=Wcid),\]
\[\text{delete Swc}(wcid=Wcid).\]

check1 Mwc(wcid=Wcid)

\[\rightarrow \text{nonvar}(Wcid) \land \sim Mwc(wcid=Wcid),\]
\[\text{write}(\text{'Work center ID does not exist in MRP II', Wcid}).\]

\[\rightarrow \text{nonvar}(Wcid) \land Mwc(wcid=Wcid).\]

check2 Prout(wcid=Wcid)

\[\rightarrow \text{nonvar}(Wcid) \land \text{nonvar}(Wcid) \land \text{EPwc (wcid=Wcid,psts=h)},\]
\[\text{write}(\text{'Work center is in use by active process plans', Wcid}).\]

\[\rightarrow \text{nonvar}(Wcid) \land \text{nonvar}(Wcid) \land \text{EPwc (wcid=Wcid,psts=r)},\]
\[\text{write}(\text{'Work center is in use by active process plans', Wcid}).\]

\[\rightarrow \text{nonvar}(Wcid) \land \sim \text{EPout (wcid=Wcid,psts=h)} \land \sim \text{EPout (wcid=Wcid,psts=r)}.\]

Figure 6.6: UDL code for the scenario "Deletion of a work center in MRP II".
6.6 Conclusions

Our implementation strategy aims at facilitating the translation between UPN and UDL (as a rule specification language) and provides us with a powerful tool to reduce the life cycle of developing knowledge bases. In addition, the same strategy can be applied in modifying existing knowledge bases, which evolve dynamically as a result of changes in existing company policies. On the other hand, if a different rule specification language was used instead of UDL, this strategy could be applied in the same way with a modified Petri nets translator. A prototype of the knowledge based system for integrating the CAD/CAPP/MRP II/SFC application systems has been developed, based on the proposed methodology. This prototype has demonstrated the feasibility of our design methodology and has won considerable attention from both industry and other related research projects.
Chapter 7

Software Development and Application

A software package has been developed to assist the design process of the knowledge base. It consists of four parts: (1) an enhanced Petri net editor, which can be used to build UPN models from the specifications of a company policy; (2) a Petri net exploder and synthesizer, which can explode an abstract net into its most detailed subnets and can synthesize separate nets (scenarios) together; (3) a Petri net analyzer, which first unfolds UPN representations into GPN representations, and then applies several analysis methods for knowledge base verification; (4) a translator, which translates the UPN models to UDL codes. These programs are coded in "C" and make use of graphical libraries of SunView. This system runs on a Unix-based Sun SPARC station. A user manual is attached in appendix B. The user should be experienced in Petri net modeling, in order to be able to create or modify subnets from company policy scenarios. Any subsequent model verification and model interpretation procedures embedded in the software are fairly straightforward and do not require any special technical skills. Most recently we used this software to model, verify and implement chemical process change control procedures at the MERCK CO. & INC. The following sections will detail the structure and functionalities of the software package and will outline its
applications.

7.1 Software Development for the Design of KBS

7.1.1 Enhanced Petri Net Editor

The Petri net editor used in the software package was adopted from the "aTrellis" hypertext system [Stotts 89] of the Department of Computer Science, University of Maryland, which focuses on browsing semantics for document structures. The original Petri net editor could only handle the creation of general Petri nets, with simple transitions representing primitive operations, and no functions attached to arcs. More entities have been added to allow for representation of UPN described in chapter 4. We first added compound transition, which represents an aggregated set of operations and need to be exploded into subnets with more detailed descriptions. Secondly, we added the capability of attaching functions to arcs, to enable modeling with Colored Petri nets. A computer screen of the Petri net editor is shown in figure 7.1 and its functionalities are described in detail in appendix B.

7.1.2 Petri Net Exploder and Synthesizer

Following the algorithms described in chapter 4, section 4.4, we added a function of exploding compound transitions into subnets automatically. This process starts from a net at the abstract level and searches for compound transitions, retrieves the subnets associated to those transitions, and links them to the abstract net through a calling subnet (calling protocol). A flow chart which describes the explosion program is shown in figure 7.2.

In addition, a new function for synthesizing separate nets (scenarios) together, through the common database places has also been added, in order to build a net
Figure 7.1: Computer screen of the Petri net editor.
Figure 7.2: Flow chart for the net explosion program.
of the entire model. A flow chart for the synthesis program is shown in figure 7.3

### 7.1.3 Petri Net Analyzer

An option for converting UPN to GPN, based on the algorithm discussed in chapter 5, section 5.2, was developed and added to the software and a flow chart for it is shown in figure 7.4.

An s-invariants generator was also implemented, which calculates the minimal set of s-invariants (linearly independent) based on the algorithm presented in [Martinez82]. The result of this generator is a file containing the description of places of each of the s-invariants. A reachability tree generator is used to obtain a
Figure 7.4: Flow chart for the abstraction and unfolding of UPN models program.
Figure 7.5: Flow chart for the consistency and conflict verification program.

set of possible final states from specified initial markings. Another option added to analyze the net is the checking on consistency and conflict from the incidence matrices of the net as discussed in chapter 5. A flow chart representing the program is shown in figure 7.5.

7.1.4 Translator Between UPN to UDL

A language translator has been implemented to translate verified UPN models into a set of UDL executable codes. This translator is based on the translation
procedure presented in chapter 6, and a flow chart representing the translation procedure is shown in figure 7.6.

7.2 Example

An example of generating a knowledge base to create and release work centers in MRP II database is presented in this section, by applying our design methodology and using this software package.
Work centers are originated in the system primarily through the MRP II module. MRP II users are responsible for establishing as well as phasing out work centers in the system, and maintaining work center data in MRP II. The specification of creating and releasing a work center in MRP II is described below.

1. MRP II user enters the basic data (Work center ID, Description, Department) to create the Work Center Record with h status and na state in MRP II.

- **checks** WC ID already exists in MRP II database

- **output** Prompt the user with an error message "WC ID already exists in MRP II database"

- **checks** WC ID not exist in MRP II database

- **creation** WC record created with h status and na state in MRP II database

2. MRP II user enters additional information (Capacity) through a modify transaction. MRP II user then releases the WC. If the additional information is not entered, the system will prompt for it during releasing the WC in MRP II.

- **checks** WC ID does not exist in MRP II database

- **output** Prompt the user with an error message "WC ID does not exist in MRP II database"

- **checks** WC ID exists in MRP II database with r status

- **output** Prompt the user with an error message "WC ID exists in MRP II database with r status"
checks WC ID exists in MRP II

WC with h status and na state exists in MRP II

All the necessary data fields are filled

WC record does not exist in CAPP

updates WC record status changed from h to r in MRP II

Skeletal WC Record automatically created in CAPP with

w status

Based on this specification, which is just a part of the scenario "Creation of a work center via MRP II", UPN models are generated by using the UPN editor as shown in figure 7.7, which represents the UPN model at the abstract level, in figure 7.8, which represents the subnet of "Creation of a work center in MRP II", and in figure 7.9, which represents the subnet of "Release of a work center in MRP II".

An explosion of the abstract net is done, and the resulted net is then converted into its equivalent GPN model by choosing respective options from the menu of the software. Several analysis techniques were applied as discussed in chapter 5, which detected no errors in the modeling process, or inconsistencies or redundancies in the model. By choosing the translation option, the UPN models of the subnets, "Creation of a work center in MRP II" and "Release of a work center in MRP II", are translated into UDL codes as shown in figure 7.10.

An example of running the codes within the UDL interpreter is shown in figure 7.11. First, both UDL codes, "crt.ud" and "rel.ud", were loaded with the command consult. Second, the user started creating a work center in MRP II with the command: "create(mwc(Wcid,Des,Dep,Cap,Sts,Ste))", The user was then prompted for the work center id number (123), description (lathe), and department (machining) of the work center. Third, the user starts the release of a work
Figure 7.7: UPN of scenario: "Creation of a work center via MRP II"
Figure 7.8: UPN of scenario: "Creation of a work center in MRP II".
Figure 7.9: UPN of scenario: "Release of a work center in MRP II".
Figure 7.10: UDL codes of "Creation of a work center in MRP II" and "Release of a work center in MRP II".
Figure 7.11: An example of creating and releasing a work center in MRP II.

center with the command: "release(mwc(Wcid,Des,Dep,Cap,Sts,Ste))". Figure 7.11 shows the error message, wc_not_exist, if the user enters a work center id number which does not exist in the MRP II database. Finally, the user enters the correct work center id number and provides its capacity. As a result, the work center status was changed from h to r in MRP II. In addition, a work center record was created in the CAPP database with w status.

7.3 Applications

The knowledge base design methodology was used to generate rules for the CAD/CAPP/MRP II/SFC system. The rule base holds about 5,000 rules, and
was developed earlier in this research without using colored Petri nets, but it has been evaluated using the new verification, and translation tools. In addition, it has been used for controlling chemical process change control procedures at the Merck CO. & INC [Garai 91]. In the manufacturing division of Merck, all manufacturing processes are operated in accordance with a standard documented procedure in a safe and environmentally sound manner to produce products of optimum chemical purity, with consistent physical properties. Any planned deviation from a current chemical process, beyond specified limits, requires the submission of a process change request which must be approved by a Plant Committee, following strict procedures and involving pre-approval, final approval and implementation of the change. In this application, general Petri nets were used to model the change procedure, which were then verified and implemented in "ProC" and interfaced with an Oracle database management system.
Chapter 8

Conclusions and Future work

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 various engineering 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 and provides a powerful tool to reduce the life cycle of developing new or modifying existing knowledge bases. Changes in existing knowledge bases evolve dynamically as a result of changes in existing company policies. A prototype of the knowledge based system for integrating CAD/CAPP/MRP II/SFC application systems has been developed and implemented with about 5,000 rules in it. This prototype has demonstrated the feasibility of our work and has won considerable attention from both industry and other related research projects. The major contributions of this work are the following:

- Developed a modeling tool, Updated Petri Nets (UPN), to facilitate the modeling of the knowledge base. UPN has added to the Colored Petri Nets (CPN) compound transitions to allow for a hierarchical structure
of knowledge representation and *global* places for database retrievals and updates.

- Formalized an abstraction algorithm which reduces the complexity of the equivalent GPN of our unfolded UPN model and facilitates the verification process.

- Implemented a translator to read the UPN model and produce the UDL executable codes automatically.

Regarding directions of some future work, which will continue to improve the efficiency and the analytical capabilities of the method, we propose:

- The development of new algorithms for the knowledge base verification, which should also include the consistency and conflict checks for more complicated rules than those discussed in chapter 5.

- The enhancement of the UPN representation schema with the timing of certain executions, and the incorporation of some probabilistic selection of alternative firing sequences.

- The implementation of the knowledge base with actual manufacturing application systems. This work will have to resolve the problems of complicated communication protocols and database accessibility issues. In addition, it will be necessary to examine the need for modeling some of the rules, internal to the application systems involved, and their compatibility with the external ones.
Bibliography


[Hardwick 89] M. Hardwick, and D. Spooner, "The ROSE Data Manager: Using Object Technology to Support Interactive Engineering


[Staff 83] Staff Report, "ICAM Program Overview", Materials Laboratory, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB, Ohio, 1983.


A.1 Creation of Work Centers in the CAD/CAPP/MRP II/SFC Integrated System via MRP II

A.1.1 Data specification

The basic work center data fields supported by the majority of MRP II, CAPP and SFC systems are as follows. Initially the data fields established by MRP II are described, as the majority of these are supported by all three application systems. Subsequently, the remaining CAPP instantiated data fields and SFC instantiated data fields will be described.

I.D. Number This is a unique identifying number assigned to each work center in the system. MRP II is the sole center for assigning the I.D. Numbers to new work centers, and these numbers identify the work center in CAPP and SFC as well.

Description This field is entered by MRP II users, and contains information as to the type of work center identified by that particular I.D. Number. This record is also maintained in CAPP and SFC.

Department This represents the department to which the work center is assigned. MRP II assigns the work centers to various departments, such as the machine shop, assembly line, quality inspection etc. CAPP and SFC include this field in their records.

Capacity This field represents the number of hours a particular work center is available for use in a given day. This information is established and maintained in MRP II, and transferred to SFC for scheduling purposes.

Resource Code This is a code which is inserted into the work center record, allocating a particular measure to the resource being utilized. The measure
could be man hours, machine hours etc. This is done for costing purposes.

**Rate Code** The Rate Code associates a particular dollar rate per hour, with the specified Resource Code.

**Dispatch Horizon** The Dispatch Horizon defines how many working days into the future are to be scanned, so as to provide workload to a work center.

**Effectivity Start Date** It represents the date on which a particular work center becomes active in the system. This date is determined by MRP II users, based on the introduction of the work center in the system, and the time required to set it up, so that it becomes fully operational. CAPP and SFC users maintain this information as well, in order to determine if the new work center will be ready in time to be utilized by a particular routing. (In fact, upon release of a W.O. to SFC, SFC should check the effectivity start date of work centers involved to make sure they are currently active.

**Effectivity End Date** This represents the date at which the work center is expected to become inactive in the system, due to obsolescence, or any other reason. This information is not required to be entered when establishing a new work center. It can be entered at a later time.

**Status Code** Status codes are used as before, to control the state of a work center in the system at any given time. MRP II, CAPP and SFC users can change the status of a particular work center, meaning that they can release, place hold, or delete a work center subject to the rules of the knowledge base.

**Work Center Load Profile** This field shows the capacity loads on a specific work center scheduled by SFC within the dispatch horizon of both MRP II and SFC.
Work Center State This field shows the physical state, either "operational" or "broken", of a work center used at the shop floor. Both SFC and MRP II use this field to monitor the availability of all the work centers, and resolve problems of machine breakdown.

In addition to some of the fields described above, CAPP systems normally maintain additional technical performance data in their work center files, which are essential for the generation of process plans, and the selection of work centers for specific operations by CAPP, and are accessible by SFC for routing selection, work order scheduling, and resource allocation. These data fields are as follows.

Horse Power This represents the rated horse power of a particular work center. It is usually given in HP.

Speed Range This field indicates the range of spindle speeds available in the work center. These speeds are represented in RPM.

Feed Range The range of feeds available in a work center are identified by this field. These are in mm./rev or mm./tooth.

Work Envelope This is a measure of the maximum volume of a workpiece that can be accommodated and worked upon by a given work center. This is represented in cm. cube.

Accuracy The rated accuracy of a work center is entered here. It is represented in microns.

Tool Change Time This is the time required to change tools in a particular work center.

Feed Change Time This is the time required to change feed in a given work center.
Speed Change Time This is the time required to change the speed in a particular work center.

Table Rotation Time This is the maximum time required to rotate the machining table, if applicable.

Tool Adjustment Time The time required to adjust the tool to the desired setting in the work center is represented here.

Rapid Traverse Rate This represents the rate of travel of the tool from one position of the workpiece to another, when in the raised position, i.e., when not cutting. It is expressed in mm/min.

A.1.2 Policy specification

Work centers are originated in the system primarily through the MRP II module. MRP II users are responsible for establishing as well as phasing out work centers in the system, and maintaining work center data in MRP II. Because CAPP requires detailed work center information for generating process plans, its work center files incorporate both the work center information maintained in MRP II and other detail technical information. Similarly, SFC also needs detailed work center information as in MRP II and CAPP. Additional work center information in SFC includes the state of a work center, to be able to schedule operations in the work orders generated by MRP II. Once again there is a great deal of similarity between the sets of data maintained by each application system.

Establishing New Work Centers in the system (Add): ADD via MRP II

1. MRP II user enters the basic data (WC ID, Description, Department) to create the Work Center Record with a status in MRP II
checks WC ID not exist in MRP II

2. MRP II user enters additional information (Capacity, Resource Code, Rate Code, Dispatch Horizon, and Effectivity Start Date) through modify transaction. MRP II user then releases the WC. Otherwise, if the additional information was not entered, the system will prompt for them during releasing of the WC in MRP II.

checks WC ID exists in MRP II

   WC with h status exists in MRP II
   All the necessary data fields are filled
   WC record does not exist in CAPP

updates WC record status changed from h to r in MRP II

   Skeletal WC Record automatically created in CAPP with
   w status

3. CAPP user enters additional information (Horse Power, Speed Range, Feed Range, Work Envelope, Accuracy, Tool Change Time, Feed Change Time, Speed Change Time, Table Rotation Time, Tool Adjusting Time, and Rapid Traverse Rate) through modify transaction. CAPP user then releases the WC. Otherwise, if the additional information was not entered, the system will prompt for them during releasing of the WC in CAPP.

checks WC ID exists in CAPP

   WC with w status exists in CAPP
   All the necessary data fields are filled
   WC record does not exist in SFC

updates WC record status changed from w to r in CAPP

   WC Record automatically created in SFC with h status
4. SFC user releases the WC with the work center state as av for being available

   checks WC ID exists
   WC with r status exists in MRP II and CAPP
   WC with h status exists in SFC

   updates WC record status changed from h to r in SFC,
   ans state changed from na to av in SFC and MRP II

A.1.3 Detailed policy specification

   MRP II users, have the sole authority to establish new work centers. This is
   because MRP II is the execution function in most manufacturing organizations,
   and is used for all types of purchasing and procurement activities. To establish
   a new work center in the system, MRP II users must provide the following basic
   information, in addition to assigning the identification number.

   • WC I.D.

   • Description

   • Department

   Subject to the condition that this WC I.D. does not already exist in the system,
   this work center record is established in MRP II with its status set to hold. MRP
   II users then finalize all the other work center details needed in MRP II module
   as follows:

   • Capacity

   • Resource Code

   • Rate Code
• Dispatch Horizon

• Effectivity Start Date

Then the MRP II user releases the work center.

This data may be entered separately as each data item becomes known, by using the modify transaction provided in the system model. Otherwise, if the additional information was not entered, the system will prompt for them during releasing of the WC in MRP II as described below. In addition to the data fields mentioned, effectivity end date, status code, work center state, and work center load profile are also part of the work center record in MRP II. The status code is not a user input, but is automatically updated from h for hold to r for released by the release transactions on the work center. The work center state is maintained by SFC users and is not provided at this stage. The work center load profile will be entered and maintained by SFC and updated automatically in MRP II after the work center is allocated for job scheduling.

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 with the help of system generated prompts. 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' as well as a work center record in the work center file in SFC with its status set to 'hold'. These work center records contain all the common information between MRP II and CAPP, and between MRP II and SFC.

CAPP users then input the detailed technical information regarding that work center. The following fields are required to be completed in CAPP, before the
work center can be given a released status, and made effective and ready to be used in process plans. If a particular data field does not apply to the specific work center, then 'inapplicable' will be entered automatically. However no field can be left blank.

- Horse Power
- Speed Range
- Feed Range
- Work Envelope
- Accuracy
- Tool Change Time
- Feed Change Time
- Speed Change Time
- Table Rotation Time
- Tool Adjusting Time
- Rapid Traverse Time

Similar to the MRP II, invoking the work center release transaction in CAPP triggers a set of consistency checks, which are as follows: the WC I.D. is checked to ensure that it exists in CAPP a work center file with status set to 'working'; all the required data fields should have been filled, and any data fields left out by users are requested at this stage with the help of system generated prompts. Upon the satisfaction of these checks, the work center gets a released status in CAPP and
the common data fields between CAPP and SFC are automatically copied from CAPP to SFC.

This scenario ends by releasing the work center in the SFC module. Invoking the work center release transaction in SFC triggers a consistency check: the WC I.D. is checked to ensure that it exists in the SFC work center file with status set to 'hold'; the WC status in both MRP II and CAPP are set as r to ensure all the necessary information have been provided; the current date is past the effectivity start date. The work center state is then automatically set to av in SFC and MRP II for being available. Upon the satisfaction of these checks, the work center gets a released status in SFC and the work center state is updated in MRP II.

### A.2 Place Hold on a Work Centers via MRP II

1. MRP II user starts the Hold of the WC record

   **checks** WC ID number exists
   
   WC is not on Hold already

   **updates** WC status changed to h in MRP II, CAPP and SFC
   
   All Routings using this WC will be put on Hold in MRP II, CAPP, and SFC.
   
   ALL Manufacturing using this WC will be put on Hold in SFC for reselecting routing and rescheduling

   Either MRP II or CAPP users can place a hold on a work center, if required. A hold invoked in MRP II could be due to routine preventive maintenance, or serious machine breakdowns. A hold invoked in MRP II will trigger the checks of the existence of the work center ID number and the status of this work center, and will also result in a hold on that workcenter in CAPP and SFC. In this situation, the work center should not be used for the shop floor scheduling, and the shop

148
floor scheduler will then look for alternative routings for those parts utilizing that work center. In addition, it is necessary for CAPP to stop using that work center in developing new process plans because the work center may be broken down for some serious problems and will not be available in a short time. All routings utilizing that work center are as a result automatically placed on hold in MRP II, CAPP, and SFC. All manufacturing orders utilizing that work center are as a result automatically placed on hold in SFC for reselecting routing and rescheduling. A work center can be rereleased at any time in MRP II, resulting in the rerelease of the work center record in CAPP and SFC, and of all the routings which utilized that work center and were placed on hold.

On the other hand, a work center placed on hold in CAPP is the result of complications arising out of manufacturing activities, on the shop floor, and the failure of the workcenter in question to carrying out the requirements of the prescribed operation. In this situation CAPP places the work center on hold, so as to temporarily suspend using that workcenter in its new process plans. In addition since this is a serious situation affecting quality, a hold invoked in CAPP automatically results in that workcenter being placed on hold in MRP II and SFC. As is the case with MRP II, all routings utilizing the workcenter are placed on hold. Therefore until further notice, alternate routings not utilizing the affected workcenter, are to be followed. All manufacturing orders utilizing that work center are as a result automatically placed on hold in SFC for reselecting routing and rescheduling. On review if it is decided to continue using the same workcenter, it can be rereleased. This results in the workcenter getting a released status automatically in MRP II. In addition all the routings utilizing the workcenter, which had been placed on hold, are released.

A.3 Deletion of Work Centers in the CAD/CAPP/MRP II/SFC Integrated System
via MRP II

1. MRP II user starts deleting the Work Center

checks WC ID exists in MRP II
   All the routing using this work center are not on h or r status in CAPP
   All the manufacturing orders using this work center are not on h or r
   status in MRP II
   All the manufacturing orders using this work center are not on h or r
   status in SFC

updates WC record removed from MRP II
   WC record removed from CAPP DBS if WC exists in CAPP
   WC record removed from SFC DBS if WC exists in SFC

Work centers can only be deleted via MRP II. As explained earlier, MRP II is
the execution function in most companies, being in charge of maintaining static
data regarding parts and work centers in the system. It is the sole center for
purchasing and procurement of resources, and in turn, is the function through
which equipment is phased out, or deleted from the system. As in the case of
inducting new work centers and other resources into the system, where MRP II
users act based on the recommendations of manufacturing and process planning
personnel, similarly, while deleting equipment, MRP II users take into account the
suggestions of these personnel. The decision is not that of MRP II users alone.

When the delete operation is invoked in MRP II, the following system checks
are initiated. A check is made to see that the work center being deleted, exists in
MRP II. The status of the work center is not relevant to the operation. In addition
all the routings maintained by the MRP II routings module are checked. If any
routings utilizing this work center exist, and are in the 'hold' or 'released' status
in CAPP, the operation fails, and a message to this effect is displayed. This is because work centers which are utilized by active routings, cannot be deleted. If any manufacturing order in MRP II and SFC utilizing this work center exist, the operation fails, and a message to this effect is again displayed. This is because work centers which are utilized by active orders, cannot be deleted. If the above checks are satisfied, the work center is deleted from the routings module of MRP II, CAPP and SFC.
Appendix

B
B.1 Overview

UPNEESAT, simplified as "UPN", is a Updated Petri Net Editor, Exploder, Synthesizer, Analyzer, and Translator, of a specialized version of Colored Petri Nets developed for the INSIM research project in the CIM Laboratory at the University of Maryland, College Park. Its editor is derived from the "aTrellis" hypertext system presented in [Stotts 89] and enhanced with additional modeling rules. Special functions have been added to automatically explode or synthesize nets. Several net analysis techniques have also been implemented into the software, in addition to a translator to translate UPN to UDL code.

This manual provides a description of the operation of the software for the modeling, analysis and translation of rule bases for integrated information systems. The functions performed by each of the modules, together with the input required and the output generated are presented.

This system allows system designers to perform the following tasks:

1. Modeling a set of rules in UPN form using the UPN editor.

2. Exploding abstract nets to detailed nets.


4. Analyzing the net.

5. Translating UPN to UDL code.

All programs have been coded in "C" with calls to SunView graphic library and currently run on a Sun SPARC station IPC. Users can invoke UPNEESAT by
executing the shell script "upn" with the option of giving it a name of an existing UPN as a parameter. Otherwise, the user can load an existing UPN or create a new one after executing "upn" without specifying a parameter.

**B.2 UPN Editor**

Going from top to bottom (and left to right), the editor functions are shown in figure 8.1. UPN components can be put on the canvas, by selecting the component icon from the editor menu and placing it at the desired location using the mouse. These functions include:

- **circle icon**, representing a place.
black bar icon, representing a primitive transition.

blank bar icon, representing a compound transition.

arrow icon, representing an arc, which can be used to connect places to transitions, or transitions to places.

black dot icon, representing a token, which can be put in places.

grouping icon, representing the selection of component clusters (or single components) which will be moved to new locations.

open circle/bar/arrow icon, representing the removal of a place, transition, or arc.

dot blinking icon, representing the deletion of token from places.

"N" (name) tag icon, representing the creation of or change to the name of a place or transition, and attaching a function to an arc.

fire icon, representing the firing of an enabled transition by clicking the mouse on it.

B.3 First Pulldown Menus

Major functions associated with the UPN editor can be selected from a pulldown menu (figure 8.2) with the middle button of the mouse. Options in the menu include:

Save writes out a document.

Load clears the canvas and brings in an existing net.

Paste brings in a net but does not clear first.
Figure 8.2: UPN first pulldown menu.
Pause/Resume time turns off/on the clock ticking.

Change Directory finds a document’s location.

Reachability computes the reachability graph of the Petri net. It is printed to a file named "tmpreach".

S-invariants computes the s-invariants of the Petri net. It is printed to a file named "tmpinv".

Explode PN explodes the current UPN model by merging all the subnets associated to the compound transitions in the UPN, and links nets at different levels through calling protocols.

Synthesize PN synthesizes subnets through global places, that represents the database state.

Abstract and unfold UPN to GPN makes the abstractions and unfolds the UPN model to its equivalent GPN model.

Analyze PN applies matrix analysis techniques to detect the redundant, subsumed, conflicting, or free-choice rules in the GPN model.

Translate UPN to UDL translates directly the UPN models to UDL codes.

**B.4 Second Pulldown Menus**

Additional functions associated to the UPN editor, which are used to format the windows and the screen, can be selected from another pulldown menu (figure 8.3) by holding down the right button of the mouse. Options in the menu include:

Tags Off/On turns the names of places and transitions, and the functions of arcs from visible to invisible, and vice versa.
Figure 8.3: UPN second pulldown menu.
Recenter moves the whole net around.

Zoom In allows the user to focus in on a smaller section of the net.

Zoom Out allows the user to see the whole picture of a large net model.

Refresh clears screen and redraws the net.

Undo recovers the previous change with only one backward step.

Clear removes everything from the canvas window.

Quit exits the software.

This software package is mostly menu driven and is portable to be run on any Sun workstation using sunview graphical environment.