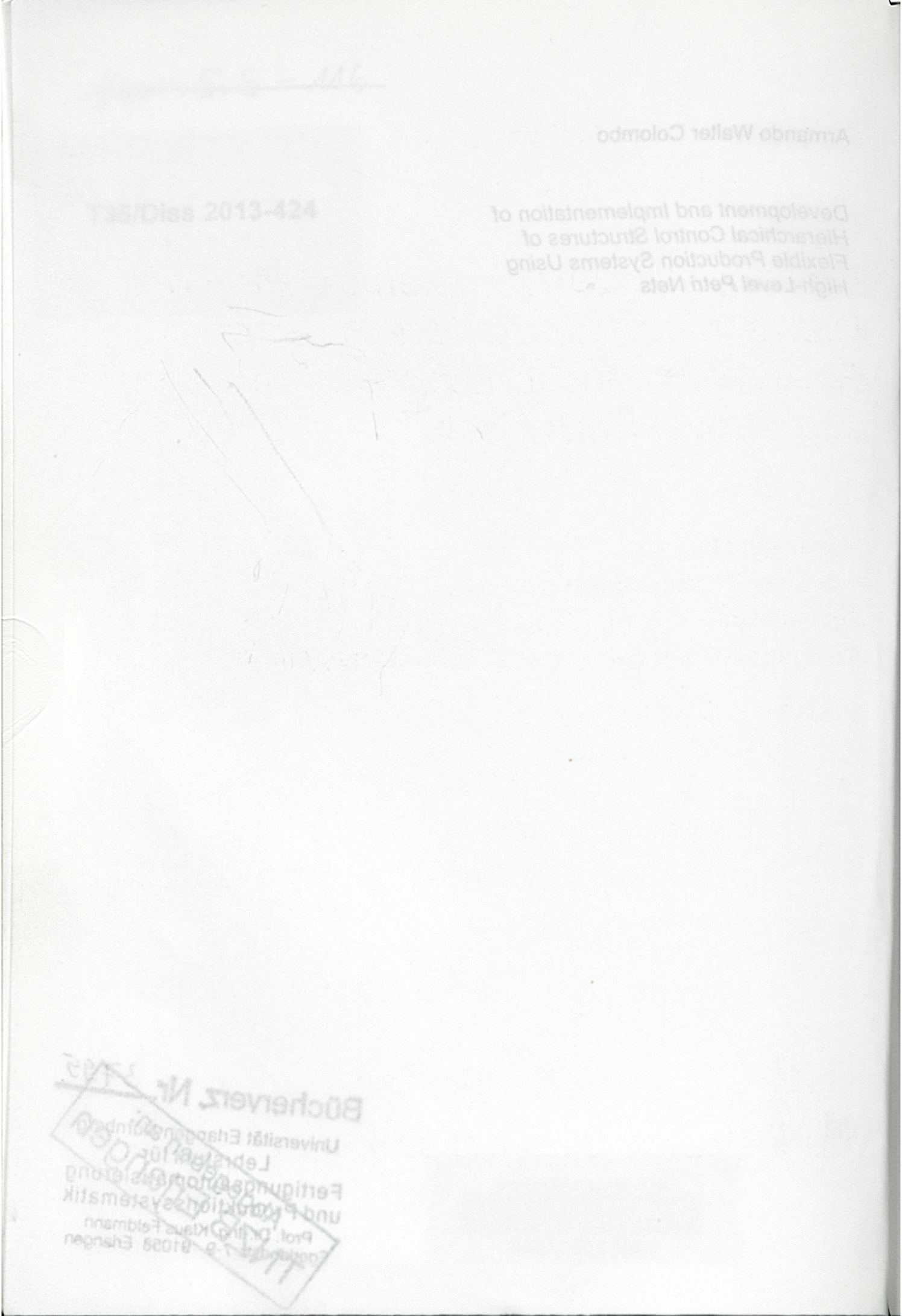
Fertigungstechnik - Erlangen

Herausgeber:
Prof. Dr.-Ing. Klaus Feldmann
Prof. Dr.-Ing. Dr. h.c. Manfred Geiger

Armando Walter Colombo

82

Development and Implementation of Hierarchical Control Structures of Flexible Production Systems Using High-Level Petri Nets



Armando Walter Colombo

Development and Implementation of Hierarchical Control Structures of Flexible Production Systems Using High-Level Petri Nets

Herausgegeben von
Professor Dr.-Ing. Klaus Feldmann,
Lehrstuhl für
Fertigungsautomatisierung und Produktionssystematik

FAPS

Meisenbach Verlag Bamberg

Als Dissertation genehmigt von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der Einreichung:	18. März 1998
Tag der Promotion:	17. Juni 1998
Dekan:	Prof. Dr. G. Herold
Berichterstatter:	Prof. Dr.-Ing. K. Feldmann
	Prof. Dr.-Ing. W. Bär



Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Colombo, Armando Walter:
Development and implementation of hierarchical control structures of flexible production systems using high level Petri nets / Armando Walter Colombo. - Bamberg : Meisenbach, 1998
(Fertigungstechnik - Erlangen ; 82)
Zugl.: Erlangen, Nürnberg, Univ., Diss., 1998
ISBN 3-87525-109-1 ISSN 1431-6226

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdrucks und der Vervielfältigung des Buches oder Teilen daraus, vorbehalten.
Kein Teil des Werkes darf ohne schrifliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung - mit Ausnahme der in den §§ 53, 54 URG ausdrücklich genannten Sonderfälle -, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© Meisenbach Verlag Bamberg 1998
Herstellung: Gruner Druck GmbH, Erlangen-Eltersdorf
Printed in Germany

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als DAAD-Stipendiat und als wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Friedrich-Alexander-Universität Erlangen-Nürnberg.

Herrn Prof. Dr.-Ing. K. Feldmann, dem Leiter dieses Lehrstuhls am Institut für Fertigungstechnik, danke ich für die wohlwollende und großzügige Förderung bei der Durchführung der Arbeit und sein mir entgegengebrachtes Vertrauen, daß mir das eigenverantwortliche Arbeiten auf diesem Forschungsgebiet ermöglichte. Mein besonderer Dank gilt ebenfalls Herrn Prof. Dr.-Ing. habil. W. Bär, Lehrstuhl für Regelungstechnik, für die Übernahme des Koreferates und Herrn Prof. Dr. F. Hofmann, Inhaber des Lehrstuhls für Informatik IV (Betriebssysteme) der Universität Erlangen-Nürnberg, dem Vorsitzenden meines Promotionsverfahrens. Beiden Professoren möchte ich darüber hinaus für die Unterstützung bei der Betreuung zahlreicher Studien- und Diplomarbeiten danken.

Prof. R. Carelli und Prof. B. Kuchen vom "Instituto de Automatica" (Universität San Juan, Argentinien), Prof. J. Martinez und Prof. M. Silva vom "Departamento de Informatica e Ingenieria de Sistemas" (Centro Politécnico Superior, Universität Zaragoza, Spanien) und Prof. Camarinha-Matos (UNINOVA, Portugal), danke ich für die fachliche Begleitung während des gesamten Entstehungszeitraums dieser Arbeit.

Ferner gilt mein Dank den Kollegen am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik, mit denen ich die Arbeit aus verschiedenen Blickwinkeln diskutieren konnte und die mir wertvolle Ratschläge und Anregungen für das Gelingen der Arbeit gaben. Den Diplomanden und wissenschaftlichen Hilfskräften, die sich mit außergewöhnlichem Engagement im Rahmen von Studien- und Diplomarbeiten an den Forschungsaufgaben beteiligt haben, danke ich für ihre Unterstützung.

Mein herzlichster Dank gilt Frau Sonja Petkovic, die in zahlreichen Korrekturnächten mit viel sprachlichem Geschick, Verständnis und Zuspruch, einen großen Beitrag zum Gelingen dieser Arbeit erbrachte.

In privater Hinsicht gilt mein aufrichtigster Dank meinen Eltern, meinem Onkel Carlos, der Familie Rearte und der Familie Hillrichs, die mir moralisch den Rücken gestärkt haben und mir die Durchführung und den erfolgreichen Abschluß dieser Arbeit ermöglichten.

Meiner Frau Fabiola Angelica, die während dieser Zeit stets großes Verständnis zeigte und mich immer tatkräftig unterstützte sowie meinem lieben Sohn Gianfranco Paolo, der häufig während seines ersten Lebensjahres auf mich verzichten mußte, gebühren mein herzlichster Dank. Beiden widme ich diese Arbeit.

Entwurf und Implementierung hierarchischer Steuerungsstrukturen von flexiblen Produktionssystemen durch High-Level-Petrinetze

– Inhaltsverzeichnis –

1. Einleitung 1

2. Grundlagen 5

2.1 Überblick zu verschiedenen Produktionsstrukturen und Konzepten 5

2.2 Flexible Produktionssysteme 6

2.3 Ereignisorientierte Steuerungssysteme 9

2.3.1 Hierarchische Struktur 9

2.3.2 Computer basierte Implementierung 12

2.3.3 Auf Speicherprogrammierbare Steuerungen basierte Implementierung 12

2.4 Werkzeuge für Design, Modellierung und Validierung von hierarchisierten Steuerungsstrukturen 15

2.4.1 Design 15

2.4.2 Modellierung 16

2.4.3 Validierung 17

2.5 High-Level-Petrinetze 19

2.5.1 Grundlagen 19

2.5.2 Dynamisches Verhalten 20

2.5.3 Erweiterungen zusätzliche Definitionen und Annahmen 21

2.5.4 Eigenschaften und Methoden zur Analyse der Netzen 30

2.5.5 Vergleich von High-Level-Petrinetze mit anderen Werkzeugen für Design und Implementierung ereignisorientierter Steuerungssysteme 33

2.6 Abschließende Bewertung 36

3. Formale Beschreibung flexibler Produktionssysteme durch High-Level-Petrinetze 38

3.1 Abbildung flexibler Produktionssysteme durch High-Level-Petrinetze 38

3.2 Modellierung flexibler Produktionssystemen mit H-L-PN 39

3.2.1 Beschreibung der Modellierungsmethodik 40
3.2.2 Erstellen eines H-L-PN-Modells der System-Ressourcen 41
3.2.3 Erstellen des H-L-PN-Koordination–Modells zum System-Layout ... 48
3.3 Verbindung der Eigenschaften von High-Level-Petrinetzen und den Spezifikationen des Systems 53
3.3.1 Analyse der H-L-PN-basierten Modellen 54
3.3.2 Validierung von Spezifikationen der Ressourcen 54
3.3.3 Validierung von Spezifikationen des System-Layouts 61
3.3.4 Validierung von Materialfluß-Spezifikationen 65
3.4 Kombination von Modellierungs- und Validierungsansätzen für das Design flexibler Produktionssysteme 73
3.5 Abschließende Bewertung 76

4. High-Level-Petrinetz-basiertes Design hierarchischer Steuerungsstrukturen von flexiblen Produktionssystemen 77

4.1 Begründung für die Realisierung hierarchischer und verteilter Steuerungsstrukturen auf der Basis von High-Level-Petrinetzen 77
4.1.1 Flexible Produktionssystem als Prozeß–Interface 79
4.1.2 Modellierung eines H-L-PN-basiertes Koordinationssystem 80
4.2 Integration von Steuerungslogik-Funktionen in das auf H-L-PN-Spezifikationen basierte Koordinationssystem 81
4.2.1 Erweiterung der H-L-PN-Struktur für die Modellierung von Steuerungslogik-Funktionen 82
4.2.2 Erzeugen der Modelle 84
4.2.3 Validierung der Steuerungs–Spezifikationen 91
4.3 Modell- und Feature-basierte Überwachung 97
4.3.1 Modell-basierte Überwachung durch Echtzeit-Analyse des Markenspieles von H-L-PN-Koordinationsmodellen 101
4.3.2 Kombination von Echtzeit-Analyse der Netz-Struktur und des Markenspieles für die modell-basierte Überwachung 103
4.3.3 Feature- und modell-basierte Überwachung des Verhaltens von Hardware-Komponenten in flexiblen Produktionszellen 104
4.4 Abschließende Bewertung 106

5. Echtzeit-Entscheidungsebene einer hierarchischen Steuerungsstruktur flexibler Produktionssysteme 107

5.1 Notwendigkeit eines Echtzeit-Entscheidungssystems zur Steuerung von flexiblen Produktionssystemen durch H-L-PN 107

5.2 Aufteilung von gemeinsam genutzten Ressourcen und anderen Materialfluß-Spezifikationen 109

5.2.1 H-L-PN-basierte, formale Beschreibung von Problemen 110

5.2.2 Problem-Behandlung und Struktur eines Echtzeit-Entscheidungssystems 113

5.2.3 Kommunikation zwischen H-L-PN-basiertem Koordinationssystem und Echtzeit-Entscheidungssystem 116

5.2.4 Kommunikation des Echtzeit-Entscheidungssystems mit den oberen Ebenen der Steuerungshierarchie 118

5.3 Einbindung von Fehler-Erkennungs- und Fehler-Behebungstrategien in die Steuerungslogik 120

5.3.1 Zuverlässigkeit des flexiblen Produktionssystems und Erweiterung der Überwachungsfunktionen des Steuerungssystems 122

5.3.2 Erweiterung der H-L-PN-basierten Steuerungsstruktur zur Einbindung von Fehlererkennungsfunktionen 122

5.3.3 Erweiterung der Steuerungsstruktur zur Einbindung von Diagnose und Fehler-Behebung als neue Überwachungsfunktionen 128

5.4 Abschließende Bewertung 131

6. Implementierung H-L-PN-basierter Steuerungsstrukturen von flexiblen Produktionssystemen in eine PC-Umgebung 132

6.1 Graphischer Editor von H-L-PN für Steuerungsfunktionen 134

6.2 Analysewerkzeug für das Verhalten von H-L-PN 137

6.3 Implementierung der H-L-PN-Kontroller in die PC-Umgebung 139

6.3.1 Struktur des Kontrollers und H-L-PN-Algorithmus 140

6.3.2 Integration des H-L-PN-basierten Steuerungssystems in eine kinematische Simulation-Umgebung 141

6.4 Implementierung eines H-L-PN-basiertes Überwachungs- und Visualisierungs-Systems in einem 2-D Visualisierungs-Tool 145

6.4.1 Entwurf eines anwenderorientierten Überwachungs- und Visualisierungs-System 147

6.4.2 Kombination von H-L-PN-basierten Informationen und technischen Signalen für Überwachungs- und Visualisierungs-Funktionen 148

6.4.3 Beschreibung der implementierten Überwachungs- und Visualisierungs-Komponenten 153

6.5 Abschließende Bewertung 154

7. Automatische Erzeugung von SPS-Anweisungen aus einer H-L-PN-basierten Beschreibung der Steuerungslogik 155

7.1 Engineering-Werkzeug für die Erzeugung von IEC 1131-konformen Steuerungslogik 155

7.1.1 Auswahl der internationalen Norm IEC 1131 156

7.1.2 Programmierungsmodell des vorgeschlagenen Verfahrens 157

7.2 Erzeugung von ST-IEC 1131-konformen Steuerungs-Anweisungen aus einer H-L-PN-basierten Beschreibung der Steuerungslogik 159

7.2.1 Einführung in eine H-L-PN-Grammatik 159

7.2.2 Erweiterung der Grammatik für die Modellierung von Logik-Kontrollern 161

7.2.3 Kompilieren der ST-IEC 1131-3-konformen Steuerungslogik aus einer H-L-PN-basierten Beschreibung 164

7.3 Übersicht zu den Komponenten und Funktionen der implementierten Werkzeuge 168

7.4 Abschließende Bewertung 172

8. Zusammenfassung und Ausblick 174

Literaturverzeichnis 176

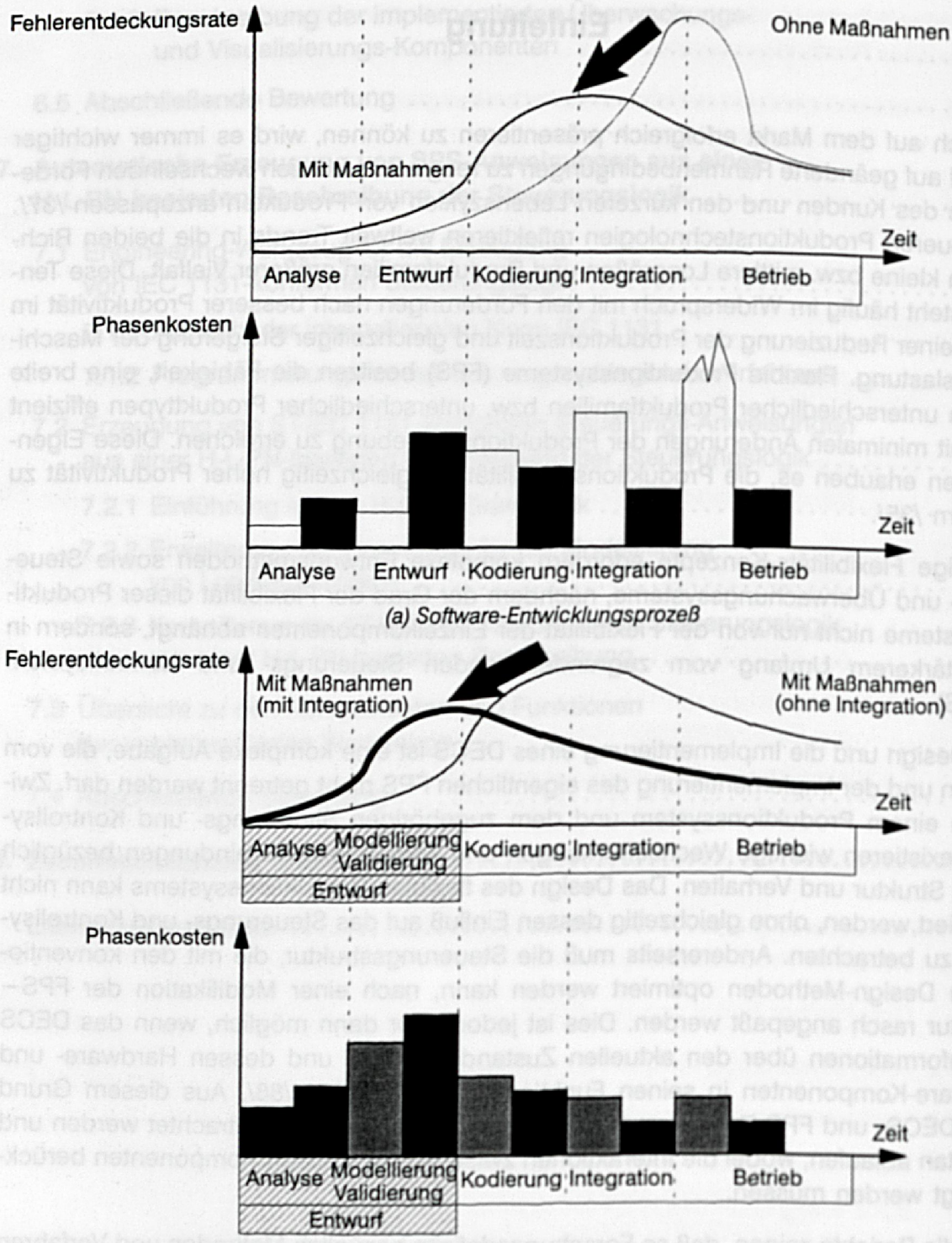
Einleitung

Um sich auf dem Markt erfolgreich präsentieren zu können, wird es immer wichtiger schnell auf geänderte Rahmenbedingungen zu reagieren und sich wechselnden Forderungen des Kunden und den kürzeren Lebenszyklen von Produkten anzupassen /37/. Die neueren Produktionstechnologien reflektieren weltweit Trends in die beiden Richtungen kleine bzw. mittlere Losgrößen und Produktfamilien größerer Vielfalt. Diese Tendenz steht häufig im Widerspruch mit den Forderungen nach besserer Produktivität im Sinne einer Reduzierung der Produktionszeit und gleichzeitiger Steigerung der Maschinenauslastung. Flexible Produktionssysteme (FPS) besitzen die Fähigkeit, eine breite Palette unterschiedlicher Produktfamilien bzw. unterschiedlicher Produkttypen effizient und mit minimalen Änderungen der Produktionsumgebung zu erreichen. Diese Eigenschaften erlauben es, die Produktionsflexibilität bei gleichzeitig hoher Produktivität zu steigern /35/.

Derartige Flexibilitäts-Konzepte erfordern komplexe Entwurfsmethoden sowie Steuerungs- und Überwachungssysteme, nachdem der Grad der Flexibilität dieser Produktionssysteme nicht nur von der Flexibilität der Einzelkomponenten abhängt, sondern in viel stärkerem Umfang vom zugrundeliegenden Steuerungs- und Kontrollsystem (DECS).

Das Design und die Implementierung eines DECS ist eine komplexe Aufgabe, die vom Design und der Implementierung des eigentlichen FPS nicht getrennt werden darf. Zwischen einem Produktionssystem und dem zugehörigen Steuerungs- und Kontrollsystem existieren wichtige Wechselwirkungen und komplizierte Verbindungen bezüglich deren Struktur und Verhalten. Das Design des flexiblen Produktionssystems kann nicht optimiert werden, ohne gleichzeitig dessen Einfluß auf das Steuerungs- und Kontrollsystem zu betrachten. Andererseits muß die Steuerungsstruktur, die mit den konventionellen Design-Methoden optimiert werden kann, nach einer Modifikation der FPS–Struktur rasch angepaßt werden. Dies ist jedoch nur dann möglich, wenn das DECS die Informationen über den aktuellen Zustand des FPS und dessen Hardware- und Software-Komponenten in seinen Funktionalitäten integriert /88/. Aus diesem Grund muß DECS- und FPS-Design und Implementierung zusammen betrachtet werden und simultan ablaufen, wobei die Interaktionen zwischen den beiden Komponenten berücksichtigt werden müssen.

Aktuelle Berichte zeigen, daß es Forschungsdefizite bezüglich Methoden und Verfahren gibt, die auf die Kostensenkung und einen detaillierten Design–/Implementierungs–Prozeß für ein reales flexibles Produktionssystem abzielen /35/, /36/, /48/, /95/, /97/, /110/, /117/. Außerdem werden Werkzeuge und Methoden für Design, Test und Implementierung eines FPS verwendet, die sich von denen der Steuerung unterscheiden. Die Implementierung des Steuerungssystems wird z.B. manuell durchgeführt, und nicht von der Modellbeschreibung des Produktionssystems abgeleitet. Zudem kann die Korrektheit bzw. Fehlerfreiheit des Design erst dann bestätigt werden wenn die Implementierung des FPS abgeschlossen ist.



(a) Software-Entwicklungsprozeß

(b) Flexibel Produktionssysteme- und Steuerungssoftware-Entwicklungsprozeß

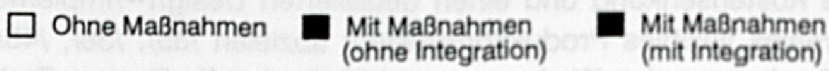


Abb. I: Tendenz bei Entwurf und Implementierung flexibler Produktionssysteme und deren Steuerung

Aufgrund der manuellen Erstellung der Steuerungssoftware getrennt von Design und Implementierung des zu steuernden Produktionssystems, nimmt der gesamte Prozeß

viel Zeit in Anspruch, führt leicht zu Mißverständnissen und Fehlern und ist folglich meist sehr kostenintensiv (Abb. I(a) nach /48/, /52/).

Vor diesem Hintergrund wird in der vorliegenden Arbeit eine formale Vorgehensweise erarbeitet, die in einem integralen Ansatz den gesamten Entwicklungsprozeß eines FPS und dessen DECS von der Anforderungsanalyse über die Modellierung und Validierung bis zur Implementierung unterstützt. Eine derartige Methodik erschließt Potentiale zur Vermeidung von hohen Kosten und lang andauernden Design-Prozessen und zur zuverlässigen Implementierung von beiden Systemen /33/. Eine Synthesemethode soll dazu vorgeschlagen werden, die es auf der Basis von systematischen Vorgehensweisen während des ganzen Designprozesses ermöglicht, formale Modelle der beiden Systemen aufzubauen. Gleichzeitig sollen dabei Spezifikationen des Produktionssystems und dessen Steuerung erhalten bleiben.

Zahlreiche mathematische Werkzeuge zur Modellierung und Steuerung flexibler Produktionssystemen sind aus der System- und Regelungstechnik-Theorie entstanden. Es ist möglich, mehrere dieser Modellen für dasselbe System anzuwenden, wobei jedes einen anderen Gesichtspunkt des Systemverhaltens darstellt /27/, /57/, /64/, /102/. Methoden zum Erstellen von Modellen eines FPS sind insbesondere dann interessant, wenn diese direkt in Modelle der Steuerungslogik transformiert und danach implementiert werden können, und zudem garantieren, daß der Anspruch auf Anwendbarkeit und Flexibilität des ganzen Systems befriedigt wird /5/, /92/. Damit ist folgende Aufgabenstellung definiert: *Integration von "Anforderungsanalyse, Modellierung und Validierung" innerhalb einer einzigen "Design Phase"* (siehe Abb. I(b)).

Kernpunkt dieser Arbeit ist die Entwicklung und Realisierung einer Engineering–Methodik, die in einem einzigen, durchgängigen Ansatz alle oben genannten Probleme in den ersten zwei Phasen des System–Entwicklungsprozesses löst. Sie basiert auf der High-Level-Petrinetz-Theorie (H-L-PN) und dem Informationsaustausch über das Auftreten von Ereignissen innerhalb der modellierten Produktionsumgebung. Das Endergebnis ist ein integriertes Entwurfswerkzeug, welches für Design und Implementierung von flexiblen Produktionssystemen und deren Steuerungsstrukturen geeignet ist.

Beim Betrachten der großen Anzahl an Interaktionen zwischen den unterschiedlichen Komponenten eines FPS und der Vielfalt der Steuerungs- und Überwachungsfunktionen, die auszuführen sind, wird festgestellt, daß DECS meistens hierarchisch und verteilt aufgebaut werden müssen. Dies führt, ausgehend von der vorgegebenen Hardware- und Software-Konfiguration eines flexiblen Produktionssystems und den Informationen über die Aufgaben und Funktionen, die im System realisiert werden müssen, zu folgender Vorgehensweise:

- Erzeugen von H-L-PN-basierten Modelle für jede Ressource (d.h. Komponenten des flexiblen Produktionssytems) und einem Koordinationsmodell der Ressourcen beim Betrachten sowohl des Konkurrenz- als auch des Kooperations-Verhaltens.
- Erzeugen einer Abbildung der Sensor–/Aktor-Schnittstellen des flexiblen Produktionssystems innerhalb einer aus dem Koordinationsmodell abgeleiteten logischen Steuerungsstruktur.

- Validierung der Spezifikationen jeder modellierten Ressourcen, der Spezifikationen des flexiblen Produktionssystem-Layouts und der Spezifikationen der logischen Steuerungsstruktur unter Verwendung formaler Analyse-Methoden (Funktionale Analyse und Lineare Algebra) in Verbindung mit den mathematischen Grundlagen der H−L−PN-Theorie, und/oder rmittels simulativer Verfahren.

Die Implementierung des entwickelten und validierten flexiblen Produktionssystems und dessen Steuerungsstruktur kann auf einer der folgenden möglichen Plattform realisiert werden:

- On-Line Steuerungs- und Überwachungsfunktionen generiert aus einem H-L-PN Interpreter (PC).
- On-Line Steuerungs- und Überwachungsfunktionen generiert aus einem H-L-PN-Logik-Kontroller (SPS).
- On-Line Steuerungs- und Überwachungsfunktionen generiert aus einer, in eine 3D-Kinematik-Simulation integrierten, H-L-PN-basierten Steuerungsstruktur.
- Off-Line Steuerungs- und Überwachungsfunktionen generiert aus einer, in eine 3D-Kinematik-Simulation integrierten, H-L-PN-basierten Steuerungsstruktur.

Die Ergebnis ist eine H-L-PN-basierte, formale Abbildung des FPS und des in ihm integrierten DECS, die vom Produktionsingenieur als eine Einheit betrachtet werden kann. Diese wird hier "Virtuelle Produktionsumgebung" genannt, die synchron mit der gesteuerten und überwachten realen Produktionsumgebung ablaufen kann.

Zusammenfassung und Ausblick

Die Faktoren "Zeit", "Flexibilität" und "Qualität" gewinnen für das Bestehen und den Erfolg von Unternehmen im harten internationalen Konkurrenzdruck zunehmend an Bedeutung. Nur mit kürzesten Entwicklungszeiten von der Produktidee bis zur Marktreife werden entscheidende Wettbewerbsvorteile gegenüber Mitbewerbern gewonnen. Die Produktion muß deshalb sowohl kundenorientiert ausgerichtet als auch in der Lage sein, auf neue Marktanforderungen flexibel und bedarfsgerecht zu reagieren. Dies kann nur durch den Einsatz flexibler Produktionssysteme (FPS) gewährleistet werden.

Der Grad der Flexibilität dieser Produktionssysteme hängt nicht nur von der Flexibilität der Einzelkomponenten ab, sondern in viel stärkerem Umfang von dem zugrundeliegenden Steuerungs- und Kontrollsystem (DECS). Aus diesem Grund erfordern derartige Flexibilitäts-Konzepte komplexe Entwurfmethoden und Steuerungs- und Überwachungssysteme.

Bisher werden zum Betrieb von flexiblen Produktionssystemen – neben CNC-, NC-, RC-Einrichtungen und Zellenrechner (PC) – speicherprogrammierbare Steuerungen (SPSen) verwendet. Der Einsatz unterschiedlicher Entwurfs- und Programmiermethoden ist bei PC-/SPS-Lösungen häufig zu beobachten und stets mit Zeitverlust verbunden. Bei einer Veränderung des FPS-Layouts oder bei einem Produktwechsel ist oft eine Um- bzw. Neuprogrammierung der Steuerungssoftware erforderlich. Deren Entwurf und Implementierung erfolgt, was den Einsatz von Werkzeugen und Methoden betrifft, weitgehend getrennt von der Planung und dem Entwurf der flexiblen Produktionssysteme selbst.

Ein neuer Ansatz, der die Integration dieser beiden getrennten Vorgehensweisen unterstützt, ist die Verwendung von High-Level Petrinetzen (H-L-PN) zu Entwurf, Modellierung, Validierung und Implementierung flexibler Produktionssysteme. H-L-PN eignen sich im Gegensatz zu anderen Verfahren weitaus besser zur Beschreibung von FPS und der auftretenden diskreten, asynchronen und nebenläufigen Prozessabläufen. Weitere Vorteile von H-L-PN ist die fundierte mathematische Theorie als Grundlage und die Möglichkeit, das Prozeßgeschehen graphisch abzubilden. Sowohl in der Planungsphase als auch in der Entwicklungs- und Implementierungsphase von FPS können H-L-PN-basierte Modelle eingesetzt werden.

Das H-L-PN-basierte Modell eines FPS bietet sich aber auch zur Steuerung des Produktionssystems an. Der Vorteil dieser Vorgehensweise liegt auf der Hand: Ist das H-L-PN-Modell eines FPS erst einmal erstellt und durch qualitative (z.B. Strukturalanalyse) sowie quantitative (z.B. Simulation zur Leistungsbewertung) Verfahren überprüft, dann kann durch Ableitung von Steuerungssignalen aus dem H-L-PN qualitativ hochwertige und fehlerfreie Steuerungslogik automatisch generiert werden. Der zeitintensive Entwicklungsschritt in wesentlichen bedingt durch die manuelle Programmierung der Steuerungsapplikation für das FPS kann drastisch reduziert werden.

Die vorliegende Arbeit beschreibt ein auf H-L-PN basierendes Verfahren zur Entwicklung von FPS, das ein neues Steuerungs- und Überwachungskonzept beinhaltet. Im

Gegensatz zu den aktuellen Lösungen kann ein FPS nach Entwurf, Modellierung und Validierung direkt durch Signalaustausch zwischen dem H-L-PN-basierten Modell und dem Produktionssystem, und durch Informationsaustausch zwischen dem H-L-PN-basierten Modell und der überlagerten Steuerungsebene (Echtzeit-Entscheidungs- bzw. Planungsebene) gesteuert werden. Die Verwendung von durchgängigen, plattformunabhängigen, konfigurierbaren und für den realen Betrieb einsetzbaren Modellen flexibler Produktionssysteme und deren Steuerungsstrukturen erschließt ein erhebliches Einsparpotential an Zeit und Kosten.

In dieser Arbeit werden zwei neue Engineering-Methoden vorgestellt. Die erste Methode unterstützt den Anwender bei Entwurf und Implementierung eines auf H-L-PN-basierten Steuerungs- und Überwachungssystems von FPS, wobei als Hardware-Plattform ein (Industrie-)PC zum Einsatz kommt. Die zweite Methode erlaubt die automatisierte Erzeugung von IEC 1131-konformem SPS-Code aus einem validierten und optimierten H-L-PN-basierten Modell eines FPS. Dabei kann der resultierende Steuerungscode direkt in die SPS bzw. einen Off-line-Simulator geladen werden.

Durch das im Rahmen dieser Arbeit entwickelte Engineering-Werkzeug kann, durch den Einsatz der H-L-PN-Theorie, der gesamte Entwicklungsprozeß, d.h. Projektierung, Programmierung und Implementierung von Steuerungssoftware für FPS, unterstützt werden.

Die in beiden Methoden verwendeten H-L-PN-basierten Modelle eignen sich aber nicht nur zur Entwicklung eines FPS und dessen Steuerung, sie können auch, durch ihren graphischen und mathematischen Charakter, das momentane systeminterne Geschehen visualisieren und dadurch zur Überwachung bzw. zur Führung von Produktionsprozessen genutzt werden. Eine Erweiterung des Steuerungskonzeptes um eine Bedienungs- und Beobachtungskomponente für FPS wurde ebenfalls entwickelt. Diese ermöglicht, neben der Visualisierung des Ablaufs von H-L-PN-Steuerungsmodellen (Markenspiel), auch die Bereitstellung von benutzerfreundlichen graphischen Darstellungen des gesteuerten Prozeßablaufes, d.h. die Integration des Menschen in das Entwicklungskonzept wird berücksichtigt (Mensch-Maschine-System).

Aufgrund der Erweiterbarkeit und des modularen Aufbaus können die in dieser Arbeit entwickelten und implementierten Steuerungsstrukturen als solide Basis für eine Reihe weiterer Entwicklungen betrachtet werden. Insbesondere stellt das implementierte Engineering-Werkzeug eine neuartige FPS-Komponente dar. Mit Hilfe dieses Werkzeugs wird eine virtuelle Abbildung der flexiblen Produktionsumgebung erstellt, welche mit dem realen FPS synchronisiert wird und damit für den Produktionsingenieur als eine Einheit anzusehen ist. Damit weist das vorgeschlagene Engineering-Werkzeug im Vergleich zu traditionellen Vorgehensweisen eine ganze Reihe von zusätzlichen Vorteilen auf: Spezifikationen können formal verifiziert werden, die Implementierung des Systems und der zugehörigen Steuerung kann automatisiert erfolgen, Test-Fälle können generiert werden, wodurch teuere Arbeitszeit beim Design des FPS eingespart werden kann. Am wichtigsten ist jedoch die Möglichkeit zur Synthese zuverlässiger Produktionssysteme (FPS/DECS) anzusehen.

Development and Implementation of Hierarchical Control Structures of Flexible Production Systems Using High-Level Petri Nets

Contents

1. Introduction 1

2. Fundamentals 5

2.1 Overview of Production Structures and Concepts 5

2.2 Flexible Production Systems 6

2.3 Discrete-Event Control Systems 9

2.3.1 Hierarchical Structure 9

2.3.2 Computer-Based Implementation 12

2.3.3 Programmable Logic Controller-Based Implementation 12

2.4 Tools for Design, Modeling and Validation of Hierarchical Control Structures 15

2.4.1 Design 15

2.4.2 Modeling 16

2.4.3 Validation 17

2.5 High-Level Petri Nets 19

2.5.1 Basic Definitions 19

2.5.2 Dynamic Behavior 20

2.5.3 Extensions and other related Definitions and Assumptions 21

2.5.4 Analysis Methods and Properties of the Nets 30

2.5.5 High-Level Petri Nets vs. other Tools in Design and Implementation of Discrete-Event Control Systems 33

2.6 Summary 36

3. Formal Specification of Flexible Production Systems using High-Level Petri Nets 38

3.1 High-Level Petri Nets and Flexible Production Systems 38

3.2 Modeling Flexible Production Systems with H-L-PN 39

3.2.1 Modeling Method 40

3.2.2 H-L-PN Model of System Resources 4
3.2.3 H-L-PN Coordination Model of the System Layout 48
3.3 High-Level Petri Net Properties and Specifications of the Systems 53
3.3.1 Analysis of the H-L-PN-Based Models 54
3.3.2 Validation of Specifications of Resources 54
3.3.3 Validation of System Layout Specifications 61
3.3.4 Validation of Material-Flow Specifications 65
3.4 Combining Modeling and Validation Approaches for Design of Flexible Production Systems 73
3.5 Summary 76

4. High-Level Petri Net-Based Design of Hierarchical Control Structures of Flexible Production Systems 77

4.1 Why a Hierarchical and Distributed Control Architecture Based on High-Level Petri Net? 77
4.1.1 Flexible Production Environment with Process Interface 79
4.1.2 H-L-PN-Based Coordination System of the Production Environment 80
4.2 Integration of Coordination Functions and Logic Control Functions using H-L-PN-Based Specifications 81
4.2.1 Extension of the Structure of H-L-PN for Modeling Logic Control Functions 82
4.2.2 Modeling 84
4.2.3 Validation of Specifications 91
4.3 Model- and Feature-Based Monitoring 97
4.3.1 Model-Based Monitoring using the Real-Time Analysis of the Token-Game of H-L-PN Coordination Control Models 101
4.3.2 Model-Based Monitoring combining Real-Time Analysis of the Net Structure and the Information obtained from the Token-Game 103
4.3.3 Feature- and Model-Based Monitoring of the Hardware Components Behavior in Flexible Production Cells Information obtained from the Structural- and Behavioral-Analysis of H-L-PN Logic Control Models 104
4.4 Summary 106

5. Real-Time Decision Level of a Hierarchical Control Architecture of Flexible Production Systems 107

5.1 Necessity of a Real-Time Decision Support System for H-L-PN Controlling of Flexible Production Systems 107

5.2 Allocation of shared Resources and Problems associated with Material-Flow Specifications 109
5.2.1 H-L-PN-Based Formal Specification of the Problems 110
5.2.2 Conflicts Treatment and Structure of a Real-Time Decision Support System 113
5.2.3 Communication between H-L-PN-Based Coordination System and RTDSS 116
5.2.4 Assistance of the Upper Levels of the DECS Hierarchy to the RTDSS Operation 118
5.3 Incorporation of Error Detection / Error Recovery Strategies in the Control Logic 120
5.3.1 Reliability of Flexible Production Systems Extension of Monitoring Functions of the DECS 122
5.3.2 Augmentation of the H-L-PN-Based Control Structures for Incorporation of Error Detection Functions 122
5.3.3 Augmentation of the DECS Structure for Incorporation of Diagnosis and Error Recovery as Supervisory Functions 128
5.4 Summary 131

6. Implementation of H-L-PN-Based Control Structures of Flexible Production Systems into a PC-Platform 132

6.1 Graphic Editor of H-L-PN for Control Purposes 134
6.2 Behavior Analyzer of H-L-PN 137
6.3 Implementation of the H-L-PN Controller into a PC-Platform 139
6.3.1 Structure of the Controller and H-L-PN Execution Algorithm 140
6.3.2 Integration of a H-L-PN-Based Controller in a Motion-Oriented Simulation Platform 141
6.4 Implementation of a H-L-PN-Based Monitoring/Visualization System using 2-D Visualization Tools 145
6.4.1 Development of a Human-Oriented Monitoring/Visualization System 147
6.4.2 Combining H-L-PN-Based Information and Technical Signals for Monitoring/Visualization Tasks 148
6.4.3 Implemented Monitoring/Visualization Component 153
6.5 Summary 154

7. Automatic Generation of PLC Code from a H-L-PN-Based Specification of Control Logic 155

7.1 Engineering Tool for Generating of IEC 1131 Control Logic Code 155

7.1.1 Why the International Standard IEC 1131 156
7.1.2 The Programming-Model of the Proposed Approach 157
7.2 Generation of ST-IEC 1131 Control Code from a H-L-PN-Based Description of the Control Logic 159
7.2.1 Introducing Grammar for H-L-PN 159
7.2.2 Extensions of the Grammar for Modelling Logic Controllers 161
7.2.3 Program for Compiling ST-IEC 1131-3 Control Logic Code according to the H-L-PN Descriptions 164
7.3 Summary of Components and Functionalities of the Implemented Tool 168
7.4 Summary 172
8. Conclusions and Outlook 174
References 176

1 Introduction

Thesis Motivation

In order to be successful in the market, it is more and more important to react quickly and opportune to alternating demands of the customers and the decreasing lifetimes of products /37/. The recent production technologies reflect a world-wide trend towards both, batches of small and medium size, and part families of increasing variety. This tendency often comes in conflict with the demand on high productivity, i.e., on production-times minimization and on simultaneous improvement of machine utilization. Potentially, flexible production systems (FPS) possess the ability to attain efficiency and versatility by producing a wide range of different product families and/or different types of a product with a minimal effort in changing the involved manufacturing environment. These characteristics make FPS also able to increase production flexibility while maintaining high productivity /35/.

However, the flexible production system's potential is not yet fully used, because of its high complexity and cost of its control systems, i.e., discrete-event control systems (DECS). There exists a lack of a methodology for quickly and economically design-implementation of DECS. This lack is considered as one of the contributing factors to the lack of widespread applications of the flexible production systems.

Design and implementation of DECS are complex tasks, which can not be separated from the design and implementation of the FPS itself. There are significant interactions and intricate relations between the production system, i.e., structure and behavior, on the one hand, and the control, on the other hand. System design can not be optimized without considering its influence upon control. In turn, control performance, which can be optimized by using conventional control design methods, can further be improved by modifying the system structure. Matching these characteristics requires the DECS to incorporate knowledge on the FPS current state and integrating hardware as well as software components /88/. Therefore, design and implementation of FPS and DECS must be integrated and performed simultaneously by considering the interactions and tradeoffs between the two.

Currently reported results show that there is not sufficient research related to a low-cost and detailed design-implementation process of practical flexible production systems, e.g., use of different tools and methods for the design, implementation, test and setting into operation /35/, /36/, /48/, /95/, /97/, /110/, /117/. For instance, the implementation of the control system is carried out manually and not derived from a model-like description of the production system. Then, the correctness of the design can only be validated after the implementation phase. Due to the manual control software production process separated from the design-implementation cycle of the production system to be controlled, the whole process is very time-consuming, presenting high rates of misunderstanding and mistakes and, as consequence, it is very expensive (see Fig. 1a, /48/, /52/).

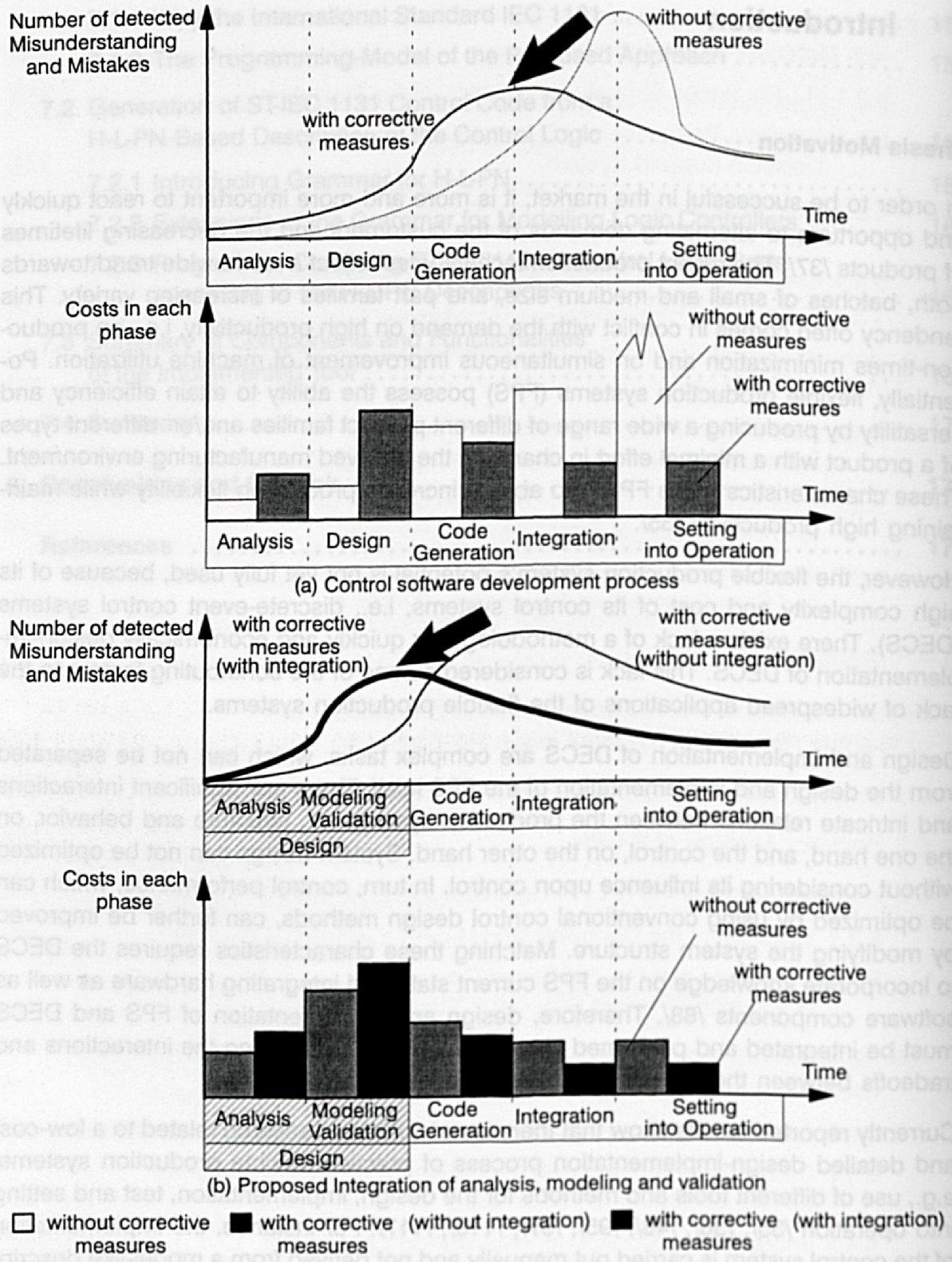


Fig. 1: Trends in the Fields of Development and Implementation of DECS

Motivated by these facts, the purpose of this work is to provide a formal methodology that covers the development life-cycle from requirements-analysis through to design-validation and implementation of flexible production systems and their discrete-event control systems in an integrated manner. Such methodology represents a way of cut-

ting down the costs and the duration of the design process and to improve a reliable implementation of both systems /33/. A synthesis method has to be proposed. It is supposed to build formal models of both systems systematically and progressively preserving the desired system's and control's specifications all along the design process.

Numerous mathematical tools for modeling and controlling flexible production systems have been emerging from the systems and control theory, and also from operation research communities. It is completely possible to employ several of these models for the same system, each representing a different point of view of the system's behavior /27/, /57/, /64/, /102/. From the operational point of view, method of constructing the control model of a flexible production system is desirable, especially if the designed model can be directly transformed into operational control logic, and it guarantees that the requirements of applicability and flexibility of the system are satisfied. Reported approaches provide, either design framework but no design methodology, or limited design policies that require trial, analysis and redesign repetitively to complete the modeling of the system and its control structure /5/, /92/.

Thesis Contribution

The greatest attention of this work is concentrated on a methodology able to help solving all above summarized problems with an unique approach. This is based on information feedback of the occurrence of events in the production environment and High-Level Petri net (H-L-PN) theory. One important assumption is now identified: *Integration of "requirement-analysis, modeling and validation" in an unique "design phase"* (see Fig. 1b), for none of these can be adequately developed independent. The final product is an integrated package suitable for design (requirements-analysis, modeling and validation) and implementation of flexible production systems and their control structures.

Taking into consideration the tremendous amount of interactions between the different components of a FPS and the variety of control and monitoring functions to be performed by its DECS, the last are mainly built in a hierarchical and distributed manner. This leads to the following proposal, where from given specifications, models are built, validated, integrated in different hardware/software platforms and finally set into operation. The steps of this methodology are in detail:

Given:

- a fixed, pre-defined set of resources (i.e., components of the flexible production system), where a resource is described by a set of specifications, port-structures for connecting it with other resources, constraints at each port-structure which describes the resources that can be connected at that port-structure, and other structural constraints, and
- a description of the desired structure (e.g., the layout of the flexible production system) and information about the set of tasks and functions which have to be

performed in each resource and in the whole system related to production objectives.

Build:

- an H-L-PN-based model for each resource of the system;
- a coordination model of resources which is obtained by taking into account both, competence and cooperation relationships, and by composing the above models of resources in a bottom-up manner;
- a mapping of the sensor/actuator interface of the flexible production cell into a discrete-event logic control structure derived from the model of each resource and also from the coordination model;
- a kinematics model with logical informations about each resource and about the whole flexible production system.

Validate:

- the specifications of each modeled resource;
- the specifications of the flexible production system;
- the specifications of the logic control structures (from components and the complete flexible production system).

This phase will be performed by means of *simulation*, and/or by using formal methods based on Functional Analysis and Linear Algebra which exploit the mathematical background of the H-L-PN.

Integrate:

- the H-L-PN-based coordination and logic control into a discrete-event control structure;
- the H-L-PN-based controller into a 3-D kinematic simulation tool.

Setting into Operation:

Four possible ways for setting the developed DECS into operation are proposed and implemented:

- on-line control and monitoring functions generated from an H-L-PN interpreter implemented in a PC-platform;
- on-line control and monitoring functions generated from an H-L-PN logic controller in a PLC-based platform;
- on-line control and monitoring functions generated from an H-L-PN-based controller embedded into a 3D-kinematic platform;
- off-line control and monitoring functions generated from an H-L-PN-based controller embedded into a 3D-kinematic platform.

As product of this phase, the H-L-PN-based formal specification of the FPS and its DECS embedded in it are treated by the production engineer as a unique entity. This is called here: *"virtual production environment"* that evolves in a synchronized manner with the controlled and monitored real production environment.

2 Fundamentals

The pressure of increasing competition together with the development of new technologies has forced widespread changes in production methodologies and control structures of the production systems /110/.

This chapter examines important fundamentals of modern production systems and related hierarchical control systems. It provides also an overview of model-based and knowledge-based methodologies as result of the use of High-Level-Petri Nets for the design and implementation of these systems.

2.1 Overview of Production Structures and Concepts

Different production structures have been developed, in order to solve the arising problems in the process of efficient product manufacturing. The manufacturing environment has evolved from intensive manual operation, where labor operated individual machines, over semiautomatic operation where the machines were able to perform a few steps in automatic sequence, to a high degree of automation making extensive use of computers, and flexible and automated equipment /89/.

The oldest production-line organization, optimized for mass production, is driven by a conveyor and the workers have to operate synchronously each doing a repetitive task. Such systems can only produce a unique kind of product. The optimization of synchronous systems consists of decomposing all the work done into a set of operations that have exactly the same duration /102/.

In the *flow shop* jobs undergo the same sequence of operations. Volumes are large and production is very efficient. Such systems can only produce a unique family of products that slightly differ from each other. Highly automated versions of the *flow shop* result in *transfer lines* (or assembly lines). The drawback of this production scheme lies in the fact that the conversion to another product is a time-consuming task.

In the *job shop* there is no notion of production line, rather for each product a *production route* is defined. This route describes a sequence of machine operations which is not restricted by the physical layout of the machines. Such systems can handle any number of product families and are limited only by the set of operations that the machines can perform. Its operation is asynchronous. It is more versatile and flexible kind of system than the *flow shop*, unfortunately it is also known to be the less efficient because either a large number of machines remain idle most of the time, or the in-process inventory is very large (completion of the products are unpredictable because they may remain a very long time in intermediary inventories) /102/.

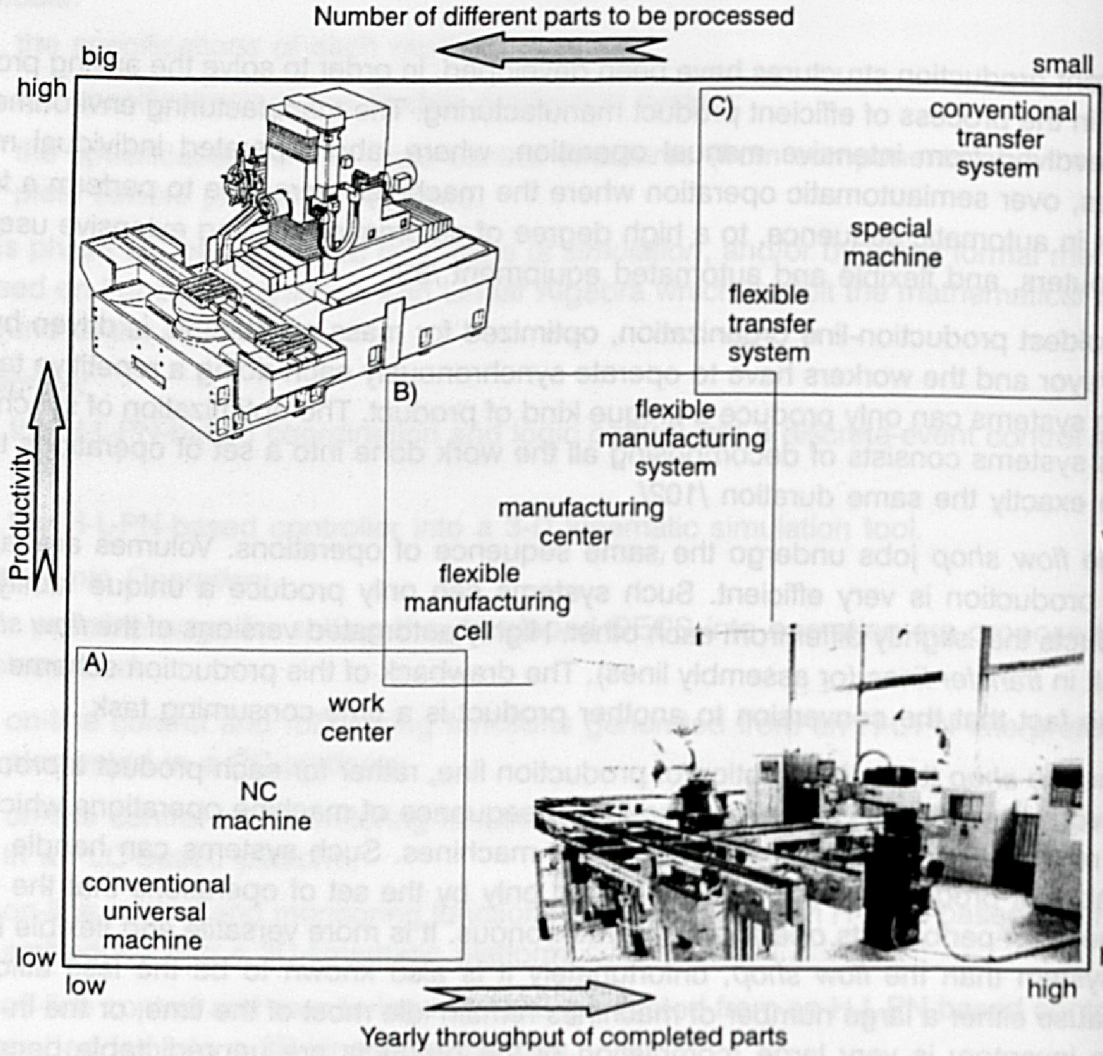
Flexible production systems (FPS) are an attempt to achieve the advantages of both the *job shop* and the *flow shop*. It reconciles the efficiency of well-balanced, machine-paced transfer lines, while utilizing the flexibility that *job shops* have, to simultaneously

machine multiple types, in order to satisfy a versatile demand at low cost /6/, /27/ /108/.

2.2 Flexible Production Systems

Compared to traditional production methods, flexible production systems have the ability to produce a family of parts simultaneously with reduced finished and in-process parts inventories, but it also responses faster to changes in demand requirements /62/ (see Fig. 2).

However, from an economic point of view, the use of a flexible production system is reasonable only within a restricted range of parameters such as productivity, number of different parts to be processed, and flexibility, among others.



Remark: A) Production of each machine; B) Flexible concept; C) Non flexible concept

Fig. 2: Application Fields of distinguished Manufacturing Concepts

Three kinds of *flexibility* are principally necessary for the operation of a FPS:

- *Long term flexibility (product flexibility)*
  The feasibility of introducing new product families in the production system during its operation with minimal modifications in the production environment, including the ability to reconfigure the system to handle the production of different products.
- *Short term flexibility (setup flexibility)*
  The possibility of handling concurrently a large variety of product families at a given time in the system with no modification in the production equipment.
- *Routing flexibility*
  The ability to route parts through the system in a dynamic fashion taking into account machine breakdowns, required tooling, etc.

In order to meet the above described requirements, a flexible production system, whether simple or complex, consists of:

- *A set of flexible resources (i.e., human operators, numerical controlled machines, robots)*
  These are capable of performing various operations on a random sequence of parts with negligible change over time from one part to the next. Flexible machines, for example, have an automatic tool storage-retrieval system, and machining programs can be down-loaded /102/. The flexibility of each resource allows the choice of one or more of them for performing each operation and also continuing the production even when one of them is out of service because of failure or maintenance.
- *An automatic transport system*
  The task of a FPS is to carry out a production plan consisting of a list of products to be processed with regard to a predefined working routing. A big requirement is that a sophisticated transportation structure allows a flexible flow of parts (i.e., raw material, tools, products) with alternate routes. This is because, in absence of a physical production line, the layout of the system does not correspond to the sequence of resource utilizations. Any location on the *shop floor* has to be reachable from any other one.
- *A sophisticated decision making system (controller)*
  A flexible production system can be straightforwardly defined as a set of resources which cooperates or competes for performing a set of activities in order to obtain a set of products. A very important activity is then the information processing necessary for production: at each instant has to be decided *"What has to be done?"*, *"When and on which resource?"*. The production system's degree of flexibility will not only be conditioned by the flexibility of its elements (workstations, storage, handling and transport systems, etc.) but will also depend fundamentally on the integrated control system /71/. The flexible production system would be of little use without a suitable control system. The last one has to allow the introduction of new family products with their routes, tolerate machine disruptions, optimize machine utilization, manage the material- and information-flow, etc.. It is this system which has to organize the production and to schedule and synchronize the resource utilization /102/.

As the target of high flexibility has to be maintained, it is required the decomposition of flexible production systems into "basic units" physically and logically identified /90/. A large number of typical production system components can be selected, analyzed from an operational point of view and then structured into *flexible production cells (FPCs).*

Flexible Production Cell

A flexible production cell can be defined as an elementary flexible production system consisting of a flexible machine tool (or an assembly device, or any complex device dedicated to a complex production operation), human operators, some local storage facilities for tools and parts and some handling devices such as robots. In order to transfer parts and tools between the cell and the global production system, an automated transport system has to be incorporated as a binding component (see Fig. 3).

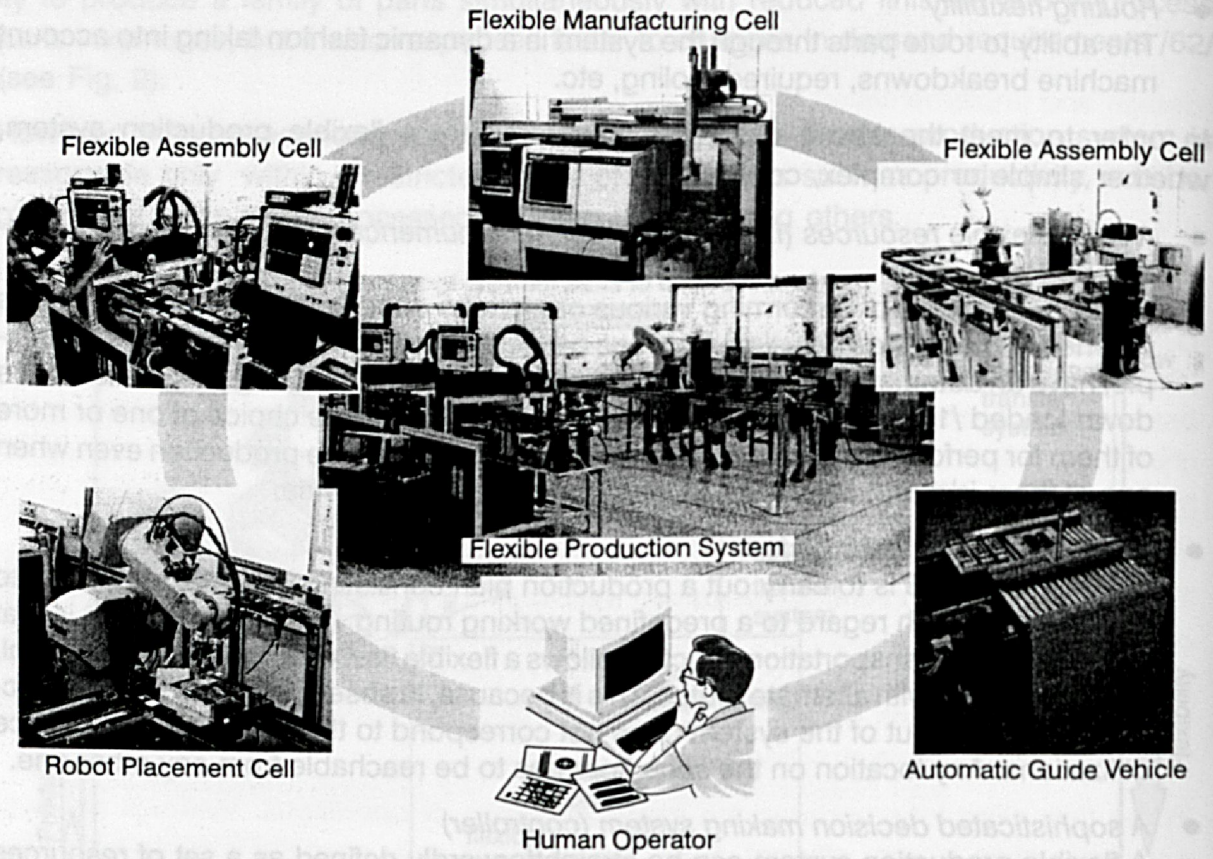


Fig. 3: Composition of a Flexible Production System

The high capital cost of a FPS/FPC means that the design and efficient operation of the system is very important. In order to meet this objectives many problems have to be solved /27/, /64/, /102/:

- an economic justification;
- selection of parts to be manufactured in the system;
- selection of machine tools;
- selection of a storage system (local buffers or central storage);
- design of material handling system;
- selection of fixtures and pallets;
- management and control of the system (how can jobs be sequenced to minimize completion time, to respond to material-flow and control sequences specifications?);
- design of computer systems and communication networks;

- layout and integration of the machines, storage system, parts handling system, computer and control systems.

This work is intended to present certain issues in development, application and implementation of techniques for solving problems related with the last three above highlighted points.

2.3 Discrete-Event Control Systems

The increased flexibility of production systems has encouraged researchers to define these complex systems as dynamic ones /27/. From this perspective, a flexible production system can be seen as a system with several independent interacting concurrent processes, exhibiting characteristics such as concurrency, conflict situations, mutual exclusion states and non-determinism. The interaction between the processes occurs in accordance with the abrupt occurrence of events and asynchronously (event-driven instead clock-driven). Typically, each independent process is split into several operations. The execution of each operation is conditioned on the satisfaction of a finite set of non-deterministic events, i.e., logical preconditions that may occur spontaneously. Upon the execution of any such operation, a new set of logical conditions is created. This set inhibits the execution of some operations and enables the execution of others in the system.

Because of these characteristics FPS and FPC can be classified as discrete-event dynamic systems (DEDS). The behavior of these systems is very difficult to describe using traditional control theory, which deals with systems of continuous or asynchronous discrete variables modeled by differential or difference equations. As many authors pointed out /5/, /27/, /35/, /36/, /46/, /66/, /102/, /104/, /117/, modeling, formal specification and validation problems emerge as a main issue for a more effective design and implementation of the real-time control of these kinds of systems, i.e., discrete-event control systems (DECSs).

In the sequel, flexible production systems and their control structures will be considered from a strictly discrete perspective.

2.3.1 Hierarchical Structure

Due to its complexity, to the tremendous amount of interactions among the different components and the variety of performed functions, DECS are currently built in a hierarchical and distributed manner. Several variations of such structure have been proposed in the literature /91/, /93/, /92/, /102/, /115/.

Based on some of those approaches, this work focuses on development and implementation of some of the components of a DECS multi-level structure, with functions closely related to the real-time "*shop floor control*".

Fig. 4 shows a hierarchical architecture of DECSs developed and implemented at the Institute for Manufacturing Automation and Production Systems in Erlangen, to perform

control of FPC. It is coarsely made up of the following main control components: *planning, real-time decision system (scheduling and dispatching), coordination, logic control, real-time monitoring and visualization, and diagnosis.*

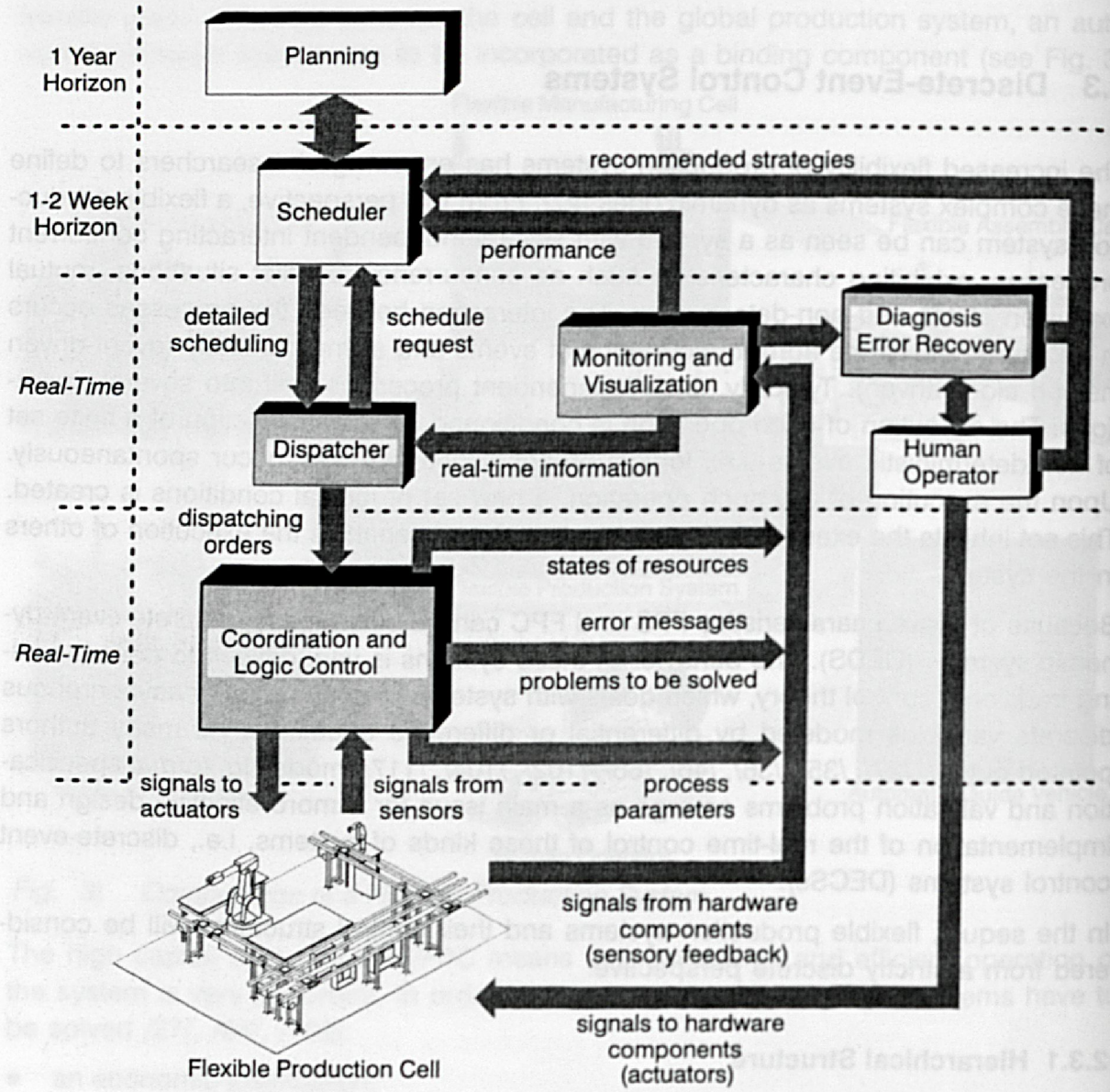


Fig. 4: Hierarchical Discrete-Event Control System for Flexible Production Cells

Each component operates within a control level and on a certain *time horizon (time scales from years and months to minutes and seconds on the shop floor)* and performs a different control function according to the view it has of the FPS/FPC under control /27/.

Two authors, /91/ and /102/, reported a very good description of some of the DECS components depicted in Fig. 4. In order to make this work self contained, each of them will be shortly described below.

Logic Control

The very lowest level of the hierarchy is called *Logic Control*. It implements the real-time control of the resources and interacts directly with the input- and output-modules of the process interface (sensors and actuators of the components of the production environment). Typical issues at this control level, among others, are the automatic control of the insertion of electronic components into printed circuit boards, set and reset of actuators, reading sensor signals from the process interface, etc.

Coordination Control

In the next level, *Coordination Control*, the detailed specifications of the logic operations are taken as shown. Its main function is to update the state representation of the production environment in real-time. If the operations themselves are treated as black boxes, the main issues of this control level are the synchronization, i.e., coordination of resources for performing these operations. There are two main goals, routing and allocation of cell components, close related to the coordination control tasks, in order to avoid deadlocks and to limit the effects of disruptions on cell operations.

Dispatcher

The main function of this component is to make real-time decisions for aiding the operation of the coordination control component.

Both, the coordinator and dispatcher, interact in a closed manner and they manage together the distribution of the elements on the resources of the cell in real-time.

Scheduling

The main function of this component is to produce a schedule, i.e., to decide at which date a given operation has to be dispatched (a sequence of dates for the execution of each operation on each machine). This component with the dispatcher together are considered within a whole structure called *real-time decision support system (RTDSS)*.

The combinatorial explosion of the number of alternatives for scheduling and dispatching is generally enormous in the case of a FPC, because each component can perform many kinds of operations, and for a given machine, once the operation is fixed, it is necessary to determine an order of execution. A *planning component* is necessary in order to efficiently reduce the number of these alternatives.

Monitoring and Visualization

In order to make good decisions, the real-time decision support system needs information related with the execution of the planned operation, the current state of the system i.e., of each resource and process, etc. The coordination controller contains itself more information about the processes that currently are developed in the system and about the conditions of the machines. This can be used together with the information coming out from the process interface for generating a complete knowledge base for monitoring and visualization goals. Such a function is performed by the monitoring and visualization component.

Diagnosis and Error Recovery

Based on the information, provided by the monitoring component about abnormal functioning of resources, e.g., the detection of wear and breakage of machine tools, errors and other unreliable operation conditions can be detected, evaluated and used as source of new knowledge. A diagnosis component is responsible for such function and will be used to issue both, automatic and manual, error recovery strategies.

Planning

Frequently various levels of planning are considered in order to support the tasks of the real-time decision system. In short-time planning level, information concerning the availability of raw material is used in order to determine at which time each product will be introduced in a FPC (*earliest starting time*) and at which time it has to be delivered (*due date*) - *Material Requirement Planning (MRP)*. Routing and scheduling remain important here. However, setup times become crucial. Frequently, it is also at this level that the resources (machines in particular) are allocated to the operations in order to balance their work load - *Manufacturing Resource Planning*. These and other planning functions are included in the upper levels of the hierarchical structure of Fig. 4.

2.3.2 Computer-Based Implementation

The distributed and hierarchical structure depicted in Fig. 4 can be seen as two connected logic systems: a real production environment, i.e., flexible production cell with its process interface /42/ and a virtual production environment, i.e., the components of the hierarchical and distributed DECS described above.

The hierarchy shown in Fig. 5 contains a substantial amount of functions distributed on the two named sub-systems. The first one (real production environment) provides the production functions which are performed in a synchronized manner with the evolution of the virtual production system represented and implemented in a PC-based platform.

The different PC-implemented components of the virtual production environment constitute the skeleton of the control system hierarchy. Moreover, they can be located at two hierarchical levels, i.e., the *DEC level* and the *Intelligence level*. The first one is composed of the coordination and logic control components described above. The second level has embedded the knowledges that are necessary to support the evolution of the DEC level. These two levels exchange information and signals to allow the whole sub-system running together with the real production world. The main characteristics of each component of both levels will be described in the following 6 chapters.

2.3.3 Programmable Logic Controller-Based Implementation

The hardware platform of the virtual production environment depicted in Fig. 5 can be modified and extended in order to allow the use of standard industrial control components such as Programmable Logic Controllers (PLC).

PLC are comparable in design to most computer-based control systems, but are specialized in logic-based control, designed to operate in industrial environments under many different conditions /5/, /26/, /43/, /112/.

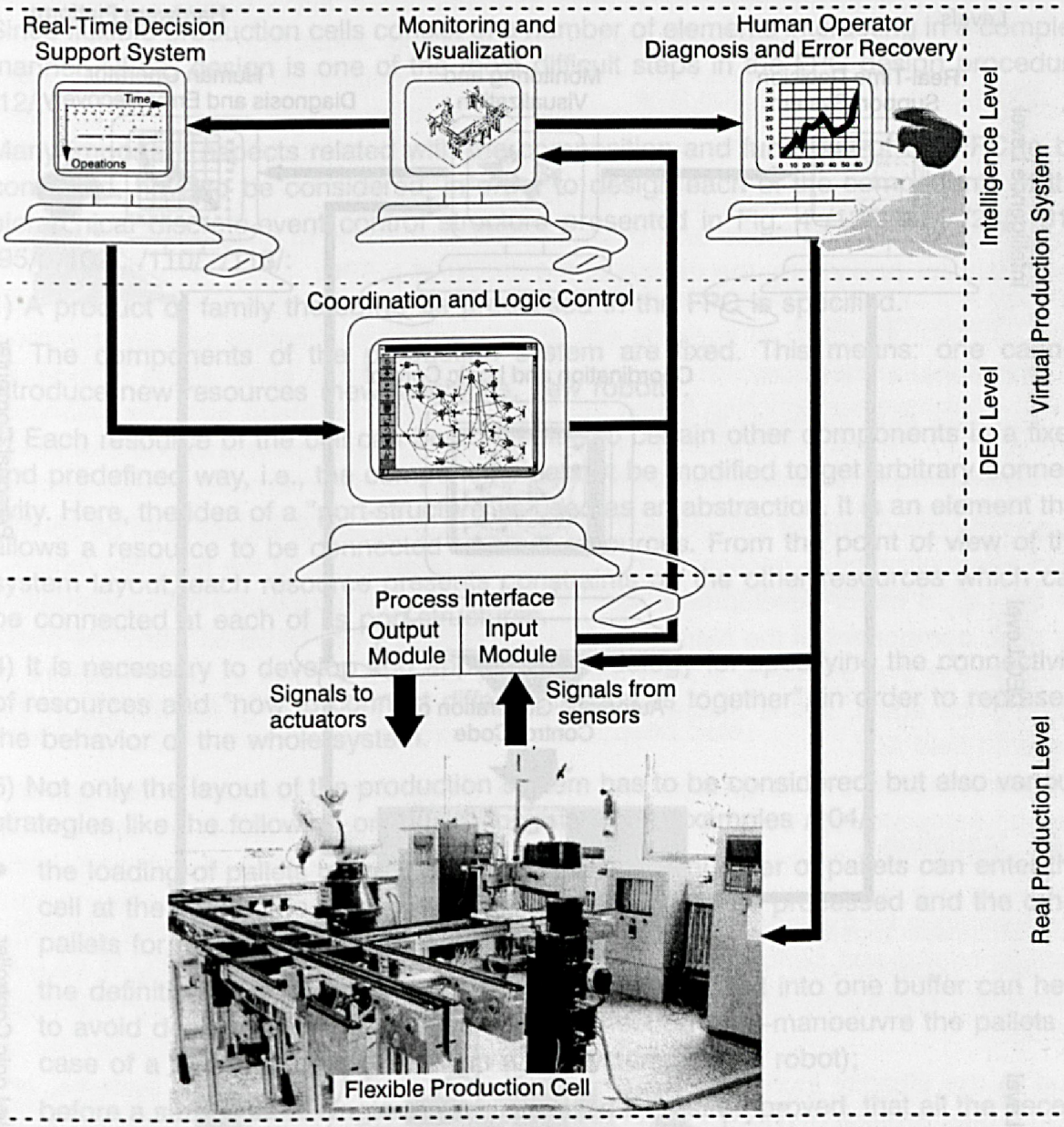


Fig. 5: PC-based Discrete-Event Control System Architecture

PLC have built-in input/output (I/O) modules linking them to the hardware components constituting the real production environment. Fig. 6 shows an architecture to implement a DECS based on the use of PLC.

It is principally composed of the same components of the PC-based architecture, however, a very important part of the virtual production environment is now replaced by a PLC.

The DEC level, i.e., the coordination and logic control, is coded on a PC and then loaded into a PLC, which acts as kernel of the whole control system. More details about this approach will be presented in chapter 7 of this work.

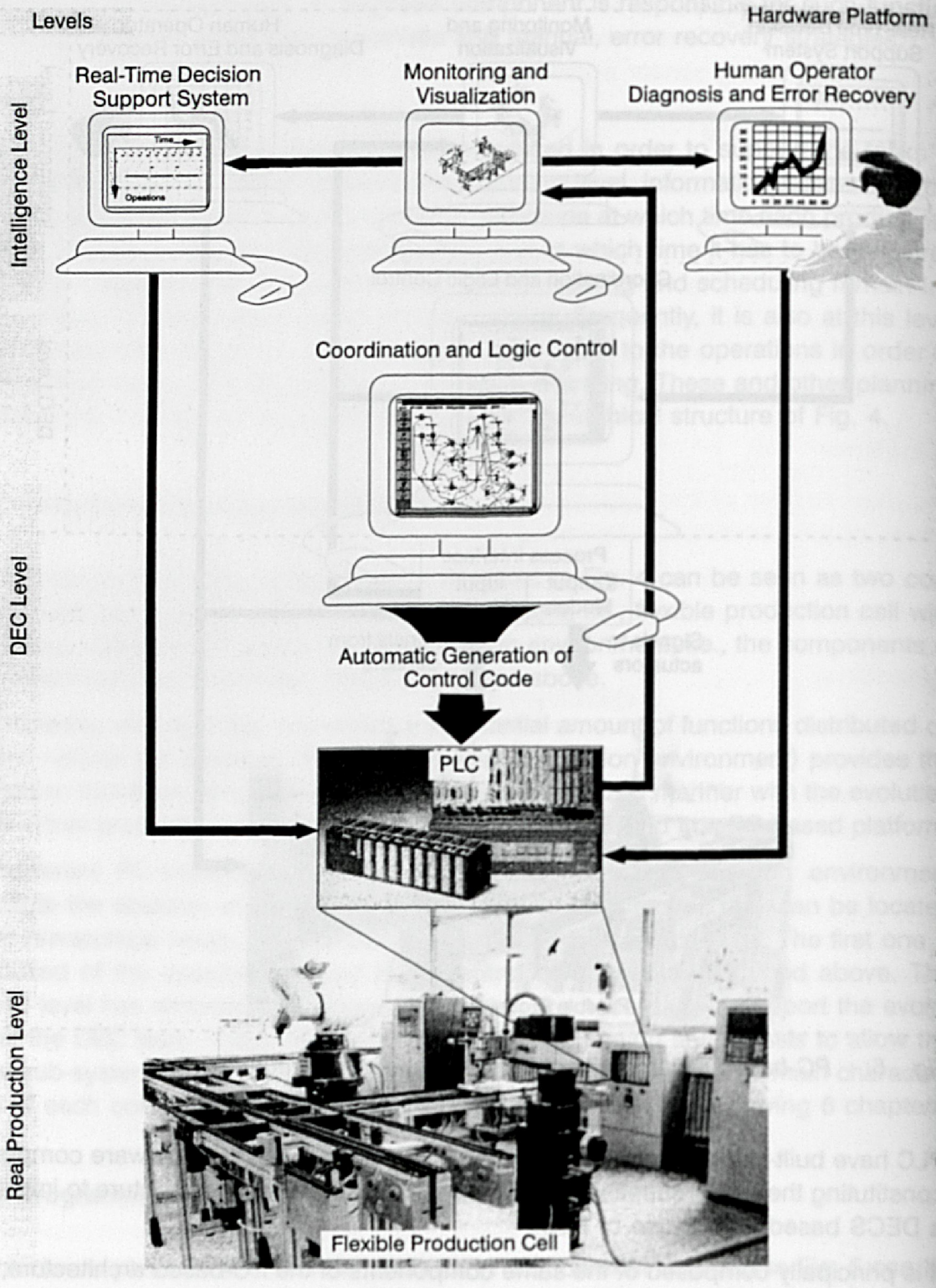


Fig. 6: PLC-based Discrete-Event Control System Architecture

2.4 Tools for Design, Modeling and Validation of Hierarchical Control Structures

2.4.1 Design

Since flexible production cells consist of a number of elements interacting in a complex manner, DECS design is one of the most difficult steps in the FPC design procedure /12/.

Many important aspects related with the composition and functions of the FPC to be controlled, have to be considered, in order to design each of the components of the hierarchical discrete-event control structure presented in Fig. 4 /12/, /21/, /28/, /91/, /95/, /102/, /110/, /118/:

1) A product or family thereof to be processed in the FPC is specified.

2) The components of the production system are fixed. This means: one cannot introduce new resources (new machines, new robots).

3) Each resource of the cell can be connected to certain other components in a fixed and predefined way, i.e., the components cannot be modified to get arbitrary connectivity. Here, the idea of a "port-structure" is used as an abstraction. It is an element that allows a resource to be connected to other resources. From the point of view of the system layout, each resource presents constraints on the other resources which can be connected at each of its port-structures.

4) It is necessary to develop and apply a methodology for specifying the connectivity of resources and "how to connect different resources together", in order to represent the behavior of the whole system.

5) Not only the layout of the production system has to be considered, but also various strategies like the following ones, just to state some examples /104/:

- the loading of pallets is only permitted, if a certain number of pallets can enter the cell at the same time (e.g., one pallet with the part to be processed and the other pallets for raw material to be assembled);
- the definition of an upper bound for pallets to be loaded into one buffer can help to avoid deadlocks (some unused space is needed to re-manoeuvre the pallets in case of a temporal deadlock within a subsystem, e.g., a robot);
- before a sequence of single actions is started it must be proved, that all the necessary resources are available in order to prevent deadlocks. In other words, these resources must be reserved exclusively.

6) There are several criteria that are used to determine control policies and strategies, like those addressed above for operating a FPC /27/. Among these are:

- Minimizing of the total time required to complete all the jobs (i.e., *minimize the makespan*)
- Minimizing the setup costs
- Meet the due date

- Minimizing of the mean time in the shop (mean flow time)
- Minimizing of machine idle time
- Minimizing of mean number of jobs in the system
- Minimizing of percentage of jobs late
- Minimizing of mean lateness of jobs
- Minimizing of mean queue time.

7) The DECS has to show the capability of integrating sensing activities with decision-making processes related to path planning, intelligent supervisory control, etc. /8/.

As a consequence of these and other aspects four kinds of knowledge are mainly necessary for performing the design of the DECS of a FPC:

- *Available resources*
- *Functional architecture*
- *Mapping from functions to resources*
- *Characteristics of a distributed architecture.*

Due to the complexity of the problem and to the large variety of situations to be considered a formalization is necessary to maintain the generality and the flexibility required for the DECS under development /90/. An appropriate methodology of representation for each component of the hierarchical control structure has to be chosen. The major point here is that the *same system* may require different models to study different issues. Once again, this is related to the lack of a unified body of knowledge in this research field /27/.

2.4.2 Modeling

The technique for the modeling and implementation of DECS presented here considers both functional and performance specifications, and properties of the FPC to be controlled, which are used as input to a design methodology for the modeling of the whole DECS.

Due to the steadily increasing complexity of the industrial production systems, and from a brief revision of referenced works, it is clear that Petri nets are a suitable modeling, analysis and implementation tool for the design of FPCs and their DECS /11/, /25/, /28/, /47/, /55/, /102/, /111/, /117/. Petri Nets have a well-founded mathematical theory and a very good capacity to formally and graphically present certain typical relationships and to visualize certain concepts, such as: parallelism and concurrency, synchronization, resource sharing, memorizing, monitoring /74/. When Petri net models are used in industrial applications, they become highly complex and are difficult to handle. In this case, the use of High-level Petri nets, e.g., colored Petri nets (CPN), has allowed creating a compact representation of states, actions and events of the modeled systems /2/, /18/, /45/, /59/, /60/, /71/.

2.4.3 Validation

Validation analysis aims at verifying the correctness of DECS design and at verifying all initial specifications. This analysis is achieved through a description, namely, a model of the system. This model needs to include all significant information on both, FPC and its DECS operations: processing-plans, resources, layout, commands, control laws, etc., and their interrelationships. The model may result in a computational model practical for analytic validation, or in a simulation model which allows experiments to be performed onto the system model.

Basically, two kinds of analysis of system model can be made: the qualitative and the quantitative analysis.

The first one verifies the compliance of certain desirable specifications of the system's components and system's behavior such as the absence of deadlock, cyclicity, finite number of states, finite capacity and boundedness of resources, possible control sequences, etc. The quantitative analysis is often called *performance evaluation*, and it takes into account system specifications, therewith checking the system's compliance with specified performance indexes, such as: production period for a part, percentual use of a resource related to a part, manufactured parts per time unit, etc. /22/.

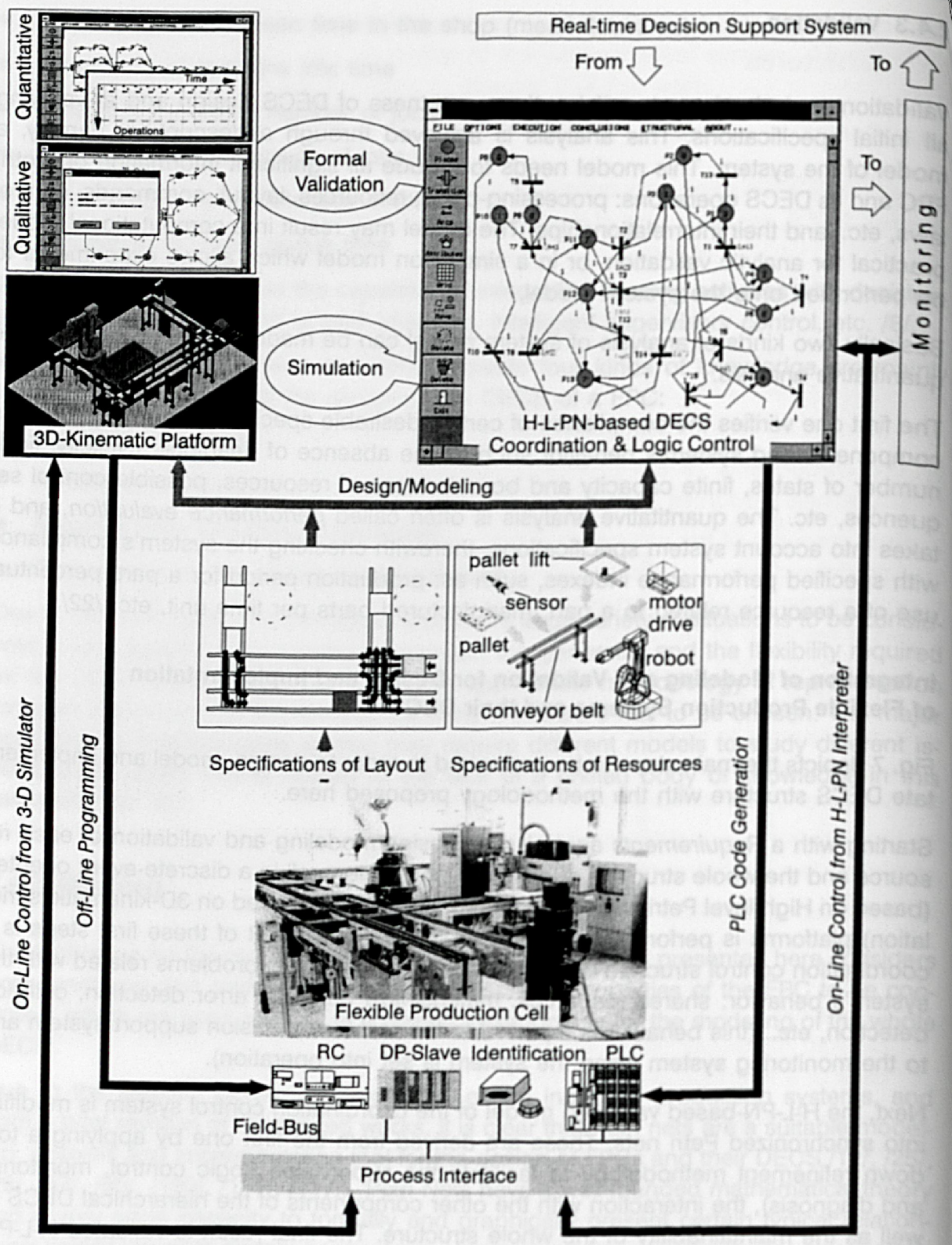
Integration of Modeling and Validation for Design and Implementation of Flexible Production Systems and their DECS

Fig. 7 depicts the main steps to be performed in order to design, model and implementate DECS structure with the methodology proposed here.

Starting with a *Requirements analysis*, the design/modeling and validation of each resource and the whole structure of a flexible production cell in a discrete-event oriented (based on High-level Petri nets) and in a motion-oriented (based on 3D-kinematic simulation) platforms is performed /10/, /17/, /41/, /84/. The result of these first steps is a coordination control structure with an identification of typical problems related with the system's behavior: shared resources, material-flow conflicts, error detection, collision detection, etc.. (this behavior is reported to the real-time decision support system and to the monitoring system when the system is set into operation).

Next, the H-L-PN-based validated model of the coordination control system is modified into synchronized Petri nets. These are derived from the first one by applying a top-down refinement methodology to facilitate the supervision (logic control, monitoring and diagnosis), the interaction with the other components of the hierarchical DECS as well as the maintainability of the whole structure. The final result is validated H-L-PN-based formal specification of the coordination control of resources of the system, and a tested logic control structures for control sequencing functions.

Finally, if the test of the logic control structure with the 3D-kinematics simulation is successful, the integration and implementation phases are carried out (more details about this approach are given in chapter 6).



Remark:
- Development
- Integration and Setting into Operation
- Communication between Component of the DECS

Fig. 7: Methodology for the Design, Modeling and Implementation of DECS

2.5 High-Level Petri Nets

Petri nets are now defined so that they can be discussed in a formal manner.

2.5.1 Basic Definitions

Generalized Petri Nets

Definition 1: A generalized Petri net (PN) is a directed, weighted, bipartite graph consisting of two types of nodes, places and transitions, where arcs are either from a place to a transition or from a transition to a place. In graphical representation, places are drawn as circles, transitions as boxes or bars. Arcs are labeled with their weights (positive integers). A marking (state) assigns to a place a non-negative integer (number of tokens).

A Petri net is defined by a 4-tuple /74/, /101/:

$$PN=\{P, T, F, W\} \tag{2.1}$$

Where:

- $P=\{p_1,p_2,...,p_m\}$ is a finite non-empty set of places,
- $T=\{t_1,t_2,...,t_n\}$ is a finite non-empty set of transitions,
- $F \subseteq \{P\times T\} \cup \{T\times P\}$ is a set of arcs (flow relation).

It is assumed that $P\cap T=\phi$ and $P\cup T\neq\phi$

- $W : F \rightarrow \mathrm{N} \setminus \{0\}$ is a weight function.
  W can also be defined by means of the input matrix E and the output matrix S.

E can be seen as function E: $P\times T \rightarrow \mathrm{N} \setminus \{0\}$. Each element of E represents the weight of the input arc from place p_i to transition t_j.
S can be seen as function S: $T\times P \rightarrow \mathrm{N} \setminus \{0\}$. Each element of S represents the weight of the output arc from transition t_i to place p_j.

Definition 2: A marking M of a PN is a function M: $P \rightarrow \mathrm{N}$, which gives the number of tokens in each place $p_i \in P$, represented by black dots. The presence or absence of tokens indicates the status of a place. The marking M can be seen as a vector, where the entry m_i corresponds to the marking of place $p_i \in P$, *i.e.*, $m_i \equiv m(p_i)$.

Definition 3: A Marked PN is defined by a 5-tuple

$$MPN=\{P, T, F, W, M_0\} \tag{2.2}$$

Where P, T, F, W were defined in (Eq. 1.1) and M_0 is the initial marking of the net, with $dim(M_0) = n$

Definition 4: For a transition t, the input set of places is defined as $\bullet t = \{p/E(p,t) \geq 0\}$. The output set of places is defined as $t\bullet = \{p/S(t,p) \geq 0\}$.

Definition 5: A transition t is said to be enabled for a marking M if, and only if $\forall\, p_i \in \bullet t$, $m(p_i) \geq E(p_i,t)$, where $m(p_i)$ denotes the number of tokens assigned by the marking M to place p_i.

A set of transitions $\tau \subseteq T$ is enabled by a marking M if, and only if $\forall\ p_i \in {}^{\bullet}\tau$, $m(p_i) \geq E(p_i,\tau)$. That is, for each place $p_i \in P$, $m(p_i)$ is equal or greater than the weight of the arcs between the place p_i an the transitions of the set τ for which p_i is an input place ($p_i \in {}^{\bullet}\tau$).

2.5.2 Dynamic Behavior

The 5-tuple of (Eq. 2.2) describes the static aspects of the net. To study its dynamics new definitions are necessary.

Definition 6 (Firing Rule): If a transition is effectively enabled, then it can be fired. After an enabled transition t_i is fired $E(p,t_i)$ tokens are removed from each of its input places ${}^{\bullet}t_i$, and $S(t_i,p)$ tokens are added to each of its output places $t^{\bullet}$.

Definition 7 (Evolution Rule): The firing of a transition t_i issues a change of the state of the net (change of the marking) which can be represented as follows:

$$M_0\ [\ t_i\ >\ M_f \tag{2.3}$$

Where M_0, M_f are the initial and final markings (states) of the net evolution.

Definition 8 (Sequence of firing): A *firing sequence* from a marking M_0 is a (possibly empty) sequence of transition sets $\theta = \tau_1\tau_2...\tau_k$ so that

$$M_0\ [\ \tau_1 > M_1\ \ [\ \tau_2 > M_2\ ....[\ \tau_k > M_k \tag{2.4}$$

It is possible to write $M_0\ [\ \theta >$ to denote that the sequence θ may be fired at M_0, and $M_0\ [\ \theta > M_k$ to denote that the firing of θ yields M_k.

Note:There are two different assumptions commonly made regarding to the number of transitions that can fire at a given instant: a) under *concurrency assumption* more than one transition of a set of enabled transitions $\tau \subseteq T$ can fire at any instant; and b) under *no concurrency assumption* only a single transition of a set of enabled transitions $\tau \subseteq T$ can fire at any instant. Under the *no concurrency assumption*, each τ_i in (Eq. 2.4) is a singleton set, and θ is a sequence of transitions.

Definition 9: A marking M is reachable in a MPN with initial marking M_0 if there exists a firing sequence θ such that $M_0\ [\ \theta > M$.

Definition 10: Given a MPN with initial marking M_0, the set of all markings reachable from M_0 (also called the *reachability set* of the net) is denoted as $\Re(M_0)$.

Definition 11: The set of all possible firing sequences from M_0 in a MPN is denoted by $\mathcal{L}(M_0)$.

Definition 12: The incidence matrix of a PN is defined as $I = S^T - E$, and it can be seen as a function $P \times T \rightarrow Z$.

Taking into consideration the definitions 7–12 the evolution of a marked Petri net can be written as a linear matrix-vector equation. Let marking M be reachable from marking M_0 by firing a sequence $\theta = \tau_1\tau_2...\tau_k$. Then the following *state transition equation* is satisfied:

$$M = M_0 + I\ .\ \Theta \tag{2.5}$$

Where $\Theta : T \rightarrow \mathrm{N}$ is a vector of non-negative integers with dimension *cardinality(T)*, called the *firing count vector* defined as

$$\Theta(t) := \Sigma_{i \in [1:k]} |\{t\} \cap \tau_i| \tag{2.6}$$

That is, $\Theta(t)$ represents the number of times transition t appears in θ. The set of markings such that there exists a vector Θ satisfying the state transition equation (Eq. 2.5) is called the *potentially reachable set* and is denoted $\mathcal{PR}(M_0)$. In general $\mathcal{PR}(M_0) \supseteq \mathcal{R}(M_0)$.

Note: As described in /74/ and /55/ respectively, the state equation (Eq. 2.5) in matrix-vector form resembles the standard state transition equation for discrete-timer linear systems, with the marking vector as the state vector, and the firing count vector as input vector. Letting M_{k+1} denote the marking after the k-th transition firing and letting Θ_k denote the k-th firing count vector, (Eq.2.5) becomes

$$M_{k+1} = M_k + I \cdot \Theta_k \tag{2.7}$$

which is strongly reminiscent of $x_{k+1} = A \cdot x_k + B \cdot u_k$ from linear systems theory. Suggestion: the existing results from linear system theory can be readily applied to the special case of Petri net dynamics. But in the Petri net dynamics is a complication, which restricts its usefulness: *only nonnegative markings are allowed during the evolution of a Petri net* (see definition 2). Nevertheless, there is a big value in viewing the Petri net dynamics from a linear algebraic perspective (see section 2.2.4). In order to generate Petri net models and to analyze their dynamics, a structural analysis theory has been developed which is mainly based on this perspective /14/, /17/, /18/, /23/, /47/, /50/, /74/, /101/.

2.5.3 Extensions and other related Definitions and Assumptions

The basic Petri net models presented in section 2.5.1 have often been enhanced and modified to serve various purposes /22/, /55/, /59/, /74/, /102/. It is important that the extensions maintain most properties of conventional Petri nets, therefore remaining an important tool for validating the model of any given system.

Four extensions of this models that will be used in this work are also described and presented here as High-Level Petri Nets (H-L-PN), namely, commutative colored Petri nets (i.e., Ordered Colored Petri nets), synchronized Petri nets for control purposes, self-adjusting synchronized Petri nets and a kind of temporized Petri nets.

Commutative Colored Petri Nets

In many cases the complexity of a PN model may be important, even for systems functionally not very complex. Under these conditions, colored Petri Nets (/45/, /59/, /60/) may bring an important contribution. The association of color sets with tokens and transitions and the definition of functions associated with the arcs of the net, allow a very concise and readable representation of complex systems with a high degree of abstraction /2/. There is a simplification of the net structure through the transfer of information to the place markings (token-colors), to the transition firings (firing- / occur-

rence-modes), and to the transformation (color-functions) made on the place markings through the functions associated with the arcs of the net.

Let us recall here, under the definition of colored Petri nets (CPN), the firing rule of transitions, the incidence matrix and the rules of the dynamical evolution of colored Petri nets /18/, /23/, /50/, /59/.

Definition 13: A CPN is a 7-tuple

$$CPN = < P, T, C, I^+, I^-, G, M_0 > \tag{2.8}$$

satisfying the following requirements:

- $P = \{ p_1, p_2, ..., p_i, ..., p_m \}$ is a finite set of places.
- $T = \{ t_1, t_2, ..., t_i, ... t_n \}$ is a finite set of transitions.
- C is the color function defined from $P \cup T$ into Ω, where Ω is a set of finite and not empty sets. An item of $C(p)$ is called a color of "p" and $C(p)$ is called the color set of "p". C attaches to each place a set of possible *token-colors* $C(p)$ and to each transition a set of possible *occurrence-colors* $C(t)$.
- I^+ (I^-) are respectively the input matrix and the output matrix defined on $P \times T$, such that $I^+(p,t)$: $C(t) \times C(p) \rightarrow \mathrm{N} \setminus \{0\}$ (i.e., a function from $C(t)$ to $Bag(C(p)) = \mathrm{N}^{C(p)}$, $\forall$ $(p,t) \in P \times T$). Elements of I^+ (I^-) are denoted, $I^+(p,(t,c_t))$, where c_t belongs to $C(t)$.
  Note: The incidence matrix I of a CPN is defined by $I = I^+ - I^-$, where
  $I(p, (t,c_t)) = I^+(p, (t,c_t)) - I^-(p, (t,c_t))$ (2.9)
  The elements of the incidence matrix can be interpreted as functions
  I: $C(p) \times C(t) \rightarrow \mathbb{Z}$
- G is the *guard function*, defined from T into expressions of type Boolean, (i.e., a predicate), such that $\forall\ t \in T$: $[Type(G(t)) = Boolean \wedge Type(Variable(G(t))) \subseteq C]$.
  $\forall \quad t \in T \wedge \quad \forall c_{ti}, c_{tj}, c_{tk} \in C(t) \Rightarrow \quad G\&_j(t) = (c_{ti} \wedge c_{tj} \quad (\neg c_{tk}) \wedge ...),\ i \neq j \neq k$
  Each $G\&_j(t)$ is a Boolean function of occurrence-colors, related to the transition t, where c_{tk} is a Boolean variable $\Rightarrow \neg c_{tk} = 1$ exactly when $c_{tk} = 0$
  According to this definition, a standard form of a guard function is described below:
  $G(t) = G\&_1(t) \vee G\&_2(t) \vee ....$ (2.10)
- M_0 is the initial marking of the net. It is a function defined on P, where
  $M_0(p)$: $C(p) \rightarrow \mathrm{N}$ (i.e., an item of Bag(C(p)), $\forall$ p $\in$ P).
  $M(p)$ and $M_0(p)$ respectively give the number of tokens of each color in the place p for the current and the initial marking. $M(p)$ and $M_0(p)$ can be seen as vectors which dimensions (cardinality) correspond to the number of places of the CPN.

Definition 14: A transition t is marking-enabled in a marking $M(p)$ for the occurrence-color $c_t \in C(t)$ if, and only if the following condition is satisfied:

$$\forall\ p \in P: \Sigma\ I^-(p, (t,c_t)) \leq M(p)$$

Definition 15: A transition t is enabled in a marking $M(p)$ iff t is marking-enabled and the corresponding guard $G(t)$ is true.

Definition 16: Two or more occurrence-colors in a transition t are concurrently enabled iff their corresponding $G\&(t)$ are true.

Definition 17: The firing of transition t for a marking $M(p)$ and a color $c_t \in C(t)$ results in a new marking $M'(p)$ defined $\forall p \in P$ by:

$$M'(p)=M(p)+I(p,(t,c_t)).\Theta \qquad (2.11)$$

Here Θ is a column vector of occurrence-colors, called the *firing counter vector,* corresponding to the occurrence sequence $\theta = t_1 t_2 ... t_k$. It can be seen as a vector of positive weighted set of transitions with dimension "*cardinality of T*". The i-th entry of Θ denotes "how many times" the transition t_i must fire regarding the respective occurrence-colors c_t to transform $M(p)$ into $M'(p)$. The term $I(p,(t,c_t)).\Theta$ denotes a matrix multiplication.

Definition 18: A set Ω_j is called "basic (standard) color domain" and its elements "color tones". Ω_j can be extended to the ring $(\Omega_j, \oplus, \odot)$, where the arithmetic functions $\oplus$ and $\odot$ are executed module s (s is the cardinality of Ω_j) /63/, /72/. For example Ω_j is the set of places of a transport system. Then, Ω_j is defined as set $\{\omega_1, \omega_2, ..., \omega_s\}$ with $s \in N$, whereby the elements $\omega_i \in \Omega_j$ refer to modeled places of the system /18/.

Note: For a basic color domain $\Omega=\{\omega_1,\omega_2,...,\omega_n\}=\{1,2,...,i,...,n\}$, the operations $\oplus$ and $\odot$ are defined as follows:

$\forall \quad a \in N, \forall \quad i \in \Omega \Rightarrow$

$\omega_i \oplus a = \omega_{i+a}$ if $\omega_i \leq \quad n_i - a$ else $a' = \omega_{i+a} - n_i$ if $\omega_i > n_{i-a}$

$\omega_i \ominus a = \omega_{i-a}$ if $\omega_i > \quad a$ else $a' = n_i - (a - \omega_i)$ if $\omega_i \leq a$

Note: The basic color domain $\Omega_k=\{<\bullet>\}$ is the set which unique element is the discolored token corresponding to the classical definition of marking /74/.

Definition 19: A complex color domain is defined as the cartesian product of two or more basic color domains.

Definition 20: The universal color domain Ω^* of the net will be the cartesian product of all basic color domains Ω_j, $j \in [1:n]$.
That is $\Omega^* = \Pi_{j \in [1:n]} \ \Omega_j = \Omega_1 \times \Omega_2 \times ... \times \Omega_n$ then $\forall \omega^* \in \Omega^* \Rightarrow \omega^* = <\omega_1,\omega_2,...,\omega_n>$
Without loss of generality, in this work it is considered that $\Omega_n=\{<\bullet>\}$.

Definition 21: The color-functions associated with the arcs of the net are the elements of the matrix I^+ (I^-). They are defined $\forall \omega^* \in \Omega^*$ and they are built from the following basic - standard - functions or their linear combination /23/, /50/: Projection functions, which select a component ω_i (color) of an item ω^*; identity functions, which select all the components of an item ω^*; successor functions, which select some successor of a component of an item; predecessor functions, which select some predecessor of a component of an item; decolored function, which transforms a colored marking in the uncolored token.

1) The projection function
$proj(k1,...,ki): \Omega^* \rightarrow (\Pi \quad \Omega_{kj}),\ j=1...i: <\omega=(\omega_1,...,\omega_n) \rightarrow \quad (\omega_{k1},...,\omega_{ki})>, \forall\ k1<k2< ...<ki$

2) The identity function
$id:\Omega^* \rightarrow \Omega^*: \quad <\omega \rightarrow \omega>$

3) The successor function
$succ(k_x):\Omega^* \rightarrow \Omega^*: \quad <\omega=(\omega_1,...,\omega_n) \rightarrow \quad (\omega_1,...,\omega_k \oplus x,...,\omega_n)>, \forall\ k<\Omega_k$

4) The predecessor function

$pred(k_x):\Omega^* \rightarrow \Omega^*$: $<\omega=(\omega_1,...,\omega_n) \rightarrow (\omega_1,...,\omega_k \ominus x,...,\omega_n)>$, $\forall\ k > 1$

5) The decolored function
$abs:\Omega^* \rightarrow \Omega_n \Rightarrow abs:\Omega^* \rightarrow \{<\bullet>\}$: $<\omega \rightarrow \omega_n=<\bullet>>$
Note: Taking into consideration the definition of $\Omega_n=\{<\bullet>\}$, the decolored function can be seen as the projection function *proj(kn)*.

6) It is also possible to build a composition of standard color functions.
a) If $succ(k_x):\Omega^* \rightarrow \Omega^*$ and $proj(kj): \Omega^* \rightarrow \Omega_k$ then $proj(kj).succ(k_x):\Omega^* \rightarrow \Omega_k \Rightarrow$
$proj(kj).succ(k_x):\Omega^*: <\omega=(\omega_1,...,\omega_n) \rightarrow (\omega_{kj} \oplus x)>$

7) Each of the defined functions can be multiplied by a constant $\mathcal{K} \in N \mid \{0\}$. This says that the colored tokens related to the function are weighted by the value of the constant.

If $proj(kj): \Omega^* \rightarrow \Omega_k$ and $\mathcal{K} \in N \mid \{0\}$ then $\mathcal{K}proj(kj): \Omega^* \rightarrow \mathcal{K}\Omega_k \Rightarrow$
$\mathcal{K}proj(kj):\Omega^*: <\omega=(\omega_1,...,\omega_n) \rightarrow (\mathcal{K}\omega_{kj})>$

For a given basic color domain K with the elements <k> it is possible to compose successor functions of that color in order to obtain the "ring function R<k>" /21/.

$$R<k>=succ(k_0)+succ(k_1)+succ(k_2)+...+succ(k_i)+...+succ(k_{n-1}) \qquad (2.12)$$

Note: Considering the above definition of basic color domains and color functions, the colored Petri net used in this work can be classified as commutative colored Petri net (OCPN) /23/. Commutative nets are a subclass of colored nets which color functions belong to a ring of commutative diagonalizable endomorphisms. Although their ability to describe models is smaller than that of colored nets, they can handle a broad range of practical problems related to flexible production systems /18/, /19/. Commutative nets include net subclasses such as regular homogeneous colored nets and ordered colored nets /50/. *This work proposes the use of Ordered Colored Petri nets (OCPN) as specification tool.*

The consideration of basic concepts of the functional analysis leads to the graphical representation of the CPN definitions shown in Fig. 8.

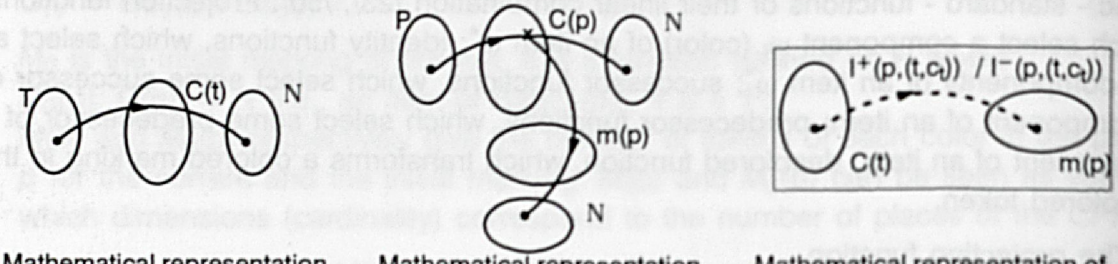


Fig. 8: Mathematical View of Functions, Marking and Firing-Colors of OCPNs

From the analysis of Fig. 8 following main conclusions are obtained:

- To each transition $t \in T$ of the net is attached a set of occurrence-colors or firing-modes $C(t)$. Each element $c_t \in C(t)$ can be mapped into the set N.

- For each place $p \in P$ of the net is attached a set of marking-colors $C(p)$. Each element $c_p \in C(p)$ can be mapped into the set N.
  To each place $p \in P$ and each color $c_p \in C(p)$ a function *"marking: m(p, c_p)"* can be defined, which gives the multiplicity $k \in$ N of this color.
- The functions attached to the arcs of the net $I^-(p,t,c_t)$ / $I^+(p,t,c_t)$ map each firing- / occurrence-color $c_t \in C(t)$ of a transition $t \in T$ into a particular marking color $c_p \in C(p)$ of the place $p \in P$.

Synchronized Petri Nets for Control Purposes

When a Petri net is used as a specification tool of a hierarchical discrete-event control system, such as presented in section 2.2, the models have to be expanded to *synchronized Petri nets.* It facilitates the supervision and the interaction of the PN-based control schema with the other components of the control structure, e.g., the controlled flexible production environment with the process interface /42/, /115/, as well as with the components of the highest level, which are the monitoring and dispatching (real-time decision system). In such a case, both the enabling and firing of transitions presented in definitions 14–17 have to be modified and extended.

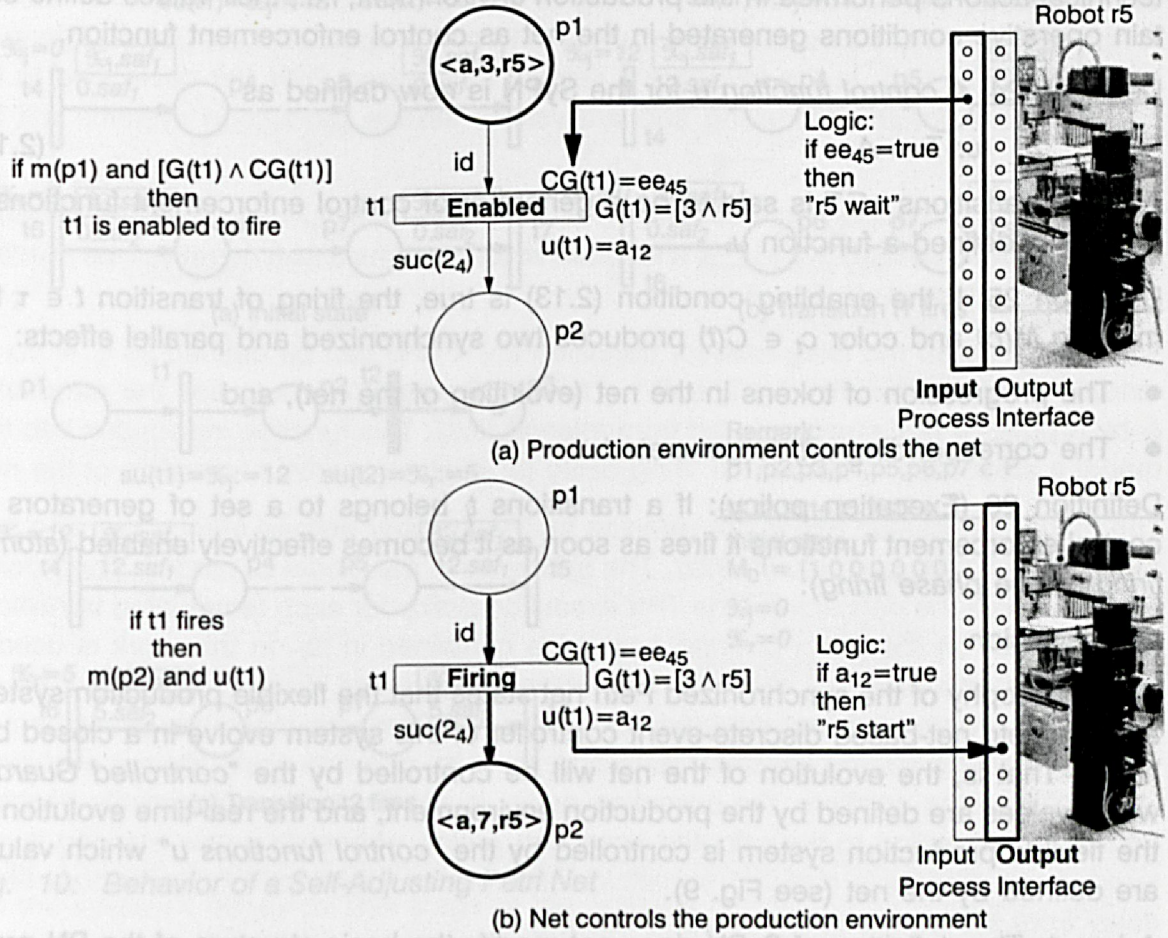


Fig. 9: Behavior of a Synchronized Petri Net

a) Extension of the enabling condition

Synchronized Petri nets (SyPN) are a class of H-L-PN with external enabling conditions called *"Controlled Guards"*.

Let $EE=\{ee_1, ee_2,...,ee_n\}, \forall\, ee_i \in \mathbb{R}+\{true, false\}, i \in \mathbb{N} \mid \{0\}$ be a set of variables that model external events, i.e., their values depend on the compliance with certain operative conditions coming out from the controlled production environment.

Definition 22: *CG* is a *controlled Guard* defined as function from *T* into expressions of type Boolean, such that $\forall\, t \in T$: $[Type(CG(t))=Boolean \wedge Type(Variable(CG(t))) \subseteq EE]$.

The enabling condition for a transition $t \in T$ (definition 15) is now expanded in order to consider the definition 23.

Definition 23: A transition $t \in T$ is enabled in a marking $M(p)$ iff t is marking-enabled and the following condition is verified:

$$[G(t) \wedge CG(t)]=true \tag{2.13}$$

b) Extension of the firing rule

Let $A=\{a_1, a_2,...,a_n\}, \forall\, a_i \in \mathbb{R}+\{true, false\}, i \in \mathbb{N} \mid \{0\}$ be a set of variables that model technical actions performed in the production environment, i.e., their values define certain operative conditions generated in the net as control enforcement function.

Definition 24: *A control function u* for the SyPN is now defined as

$$u: T \rightarrow A \tag{2.14}$$

A set of transitions $\tau \subseteq T$ is said to be a generator of control enforcement functions if $\forall\, t_i \in \tau$ is defined a function *u*.

Definition 25: If the enabling condition (2.13) is true, the firing of transition $t \in \tau$ for marking $M(p)$ and color $c_t \in C(t)$ produces two synchronized and parallel effects:

- The progression of tokens in the net (evolution of the net), and
- The corresponding $u(t)$ is generated.

Definition 26 (Execution policy): If a transitions t_i belongs to a set of generators of control enforcement functions it fires as soon as it becomes effectively enabled (*atomic firing – one phase firing*).

c) Conclusion

The philosophy of the synchronized Petri net states that the flexible production system and the Petri net-based discrete-event controller of this system evolve in a closed behavior. That is, the evolution of the net will be controlled by the *"controlled Guards"* which values are defined by the production environment, and the real-time evolution of the flexible production system is controlled by the *"control functions u"* which values are defined by the net (see Fig. 9).

Axiom 1: The definition of SyPN does not modify the basic structure of the PN presented in section 2.5.1, therefore maintaining the set of properties of the net. Also, the set of firing sequences of SyPN is a subset of the firing sequences of its non synchro-

nized PN, since the synchronization with external signals and information may inhibit certain combinations of firing transitions.

Self-Adjusting Petri Nets

They are an extension of the above presented synchronized Petri nets and were developed for modeling and control special kinds of flexible production cells, i.e., flexible robot placement systems (see chapter 6). In this kind of nets, the value of the color-functions associated with some arcs of the net (see definitions 13 and 21) is adjusted during the evolution of the net by the *control function u* associated to some transition of the same net.

Fig. 10 depicts the principal characteristics of the behavior of a self-adjusting Petri net.

Let $U=\{u_1, u_2, ..., u_n\}$ be the set of control functions and $SU \subseteq U$, $SU=\{su_1, su_2,.., su_k\}$ with $k \leq n$ the set of control functions defined to adjust some color functions on a synchronized Petri net.

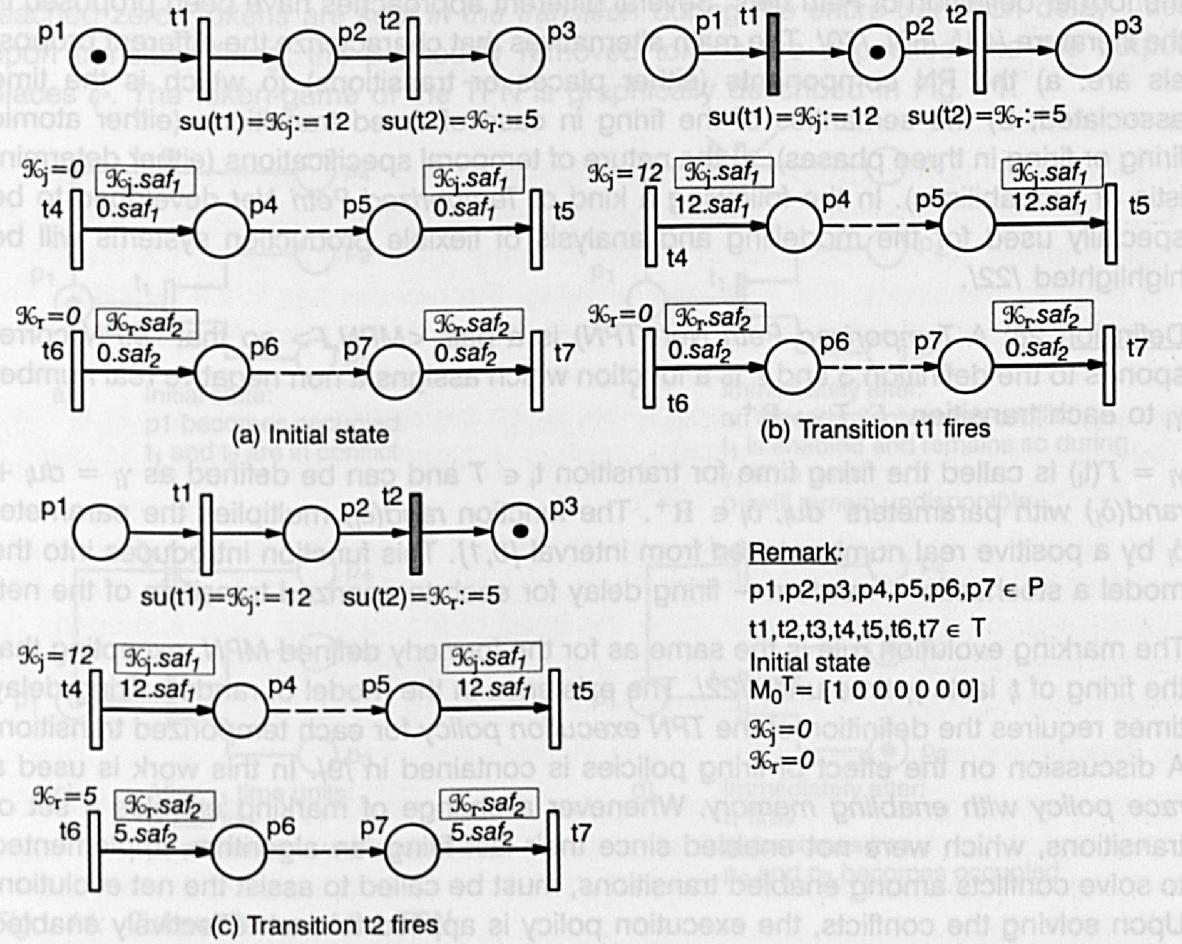


Fig. 10: Behavior of a Self-Adjusting Petri Net

Definition 27: *saf* is a self-adjusting arc function, defined as

$$\mathcal{H}_i.saf: P \times T \rightarrow N \qquad (2.15)$$

For the initial state of the net, the value of $\mathcal{G}_i$ is $\{0\}$. When the net evolves the value of $\mathcal{G}_i$ will be determined by a su_x associated with some transition t_x of it. A new value for $\mathcal{G}_i$ will be defined as soon as the transition t_x fires.

For the purpose of this work it is essential to develop Petri net-based models of flexible production systems which fulfil the essential set of properties highlighted in section 2.5.4. For this reason, in addition to the definition 27, an important feature has to be considered when modeling with self-adjusting synchronized Petri nets.

Axiom 2: In a self-adjusting Petri net there is always a pair of arcs with the same self-adjusting arc function $\mathcal{G}_i.saf$. The models must verify the so called property *conservativeness* /74/ (see section 2.5.4).

Temporized Petri Nets

They are introduced in order to extend PN modelling and analysis potentialities for the evaluation of system performance. For this purpose, the time factor is incorporated to the former definition of Petri nets. Several different approaches have been proposed in the literature /44/, /69/, /70/. The main alternatives that characterize the different proposals are: a) the PN components (either places or transitions) to which is the time associated; b) the semantics of the firing in case of timed transitions (either atomic firing or firing in three phases); c) the nature of temporal specifications (either deterministic or probabilistic). In the following a kind of *Temporized Petri Net* developed to be specially used for the modelling and analysis of flexible production systems will be highlighted /22/.

Definition 28: A *Temporized Petri Net (TPN)* is a pair $<MPN,\Gamma>$ so that *MPN* corresponds to the definition 3 and Γ is a function which assigns a non negative real number γ_i to each transition, $\Gamma: T \rightarrow \mathbb{R}^+$.

$\gamma_i = \Gamma(t_i)$ is called the firing time for transition $t_i \in T$ and can be defined as $\gamma_i = du_i + rand(\delta_i)$ with parameters du_i, $\delta_i \in \mathbb{R}^+$. The function $rand(\delta_i)$ multiplies the parameter δ_i by a positive real number taken from interval $[0,1]$. This function introduces into the model a stochastic – random – firing delay for each temporized transition of the net.

The marking evolution rule is the same as for the formerly defined *MPN*, excepting that the firing of t_i lasts γ_i time units /22/. The existence in the model of random firing delay times requires the definition of the *TPN execution policy* for each temporized transition. A discussion on the effect of firing policies is contained in /9/. In this work is used a *race policy with enabling memory*. Whenever a change of marking enables a set of transitions, which were not enabled since their last firing, an algorithm, implemented to solve conflicts among enabled transitions, must be called to assist the net evolution. Upon solving the conflicts, the execution policy is applied to each effectively enabled transition. According to the adopted execution policy, whenever a new marking is entered, each effectively enabled transition t_i samples an instance of the random firing delay from the associated $\gamma_i = du_i + rand(\delta_i)$.

Axiom 3: Transitions compete for firing. Among the set of enabled transitions, the competition is won by the transition that samples the shortest delay time.

Because of this rule, immediate transitions ($\gamma_i=0$) have priority to fire over timed transitions. If two or more enabled transitions have identical delay time, they fire together. *Enabling memory* indicates that the re-sampling is performed only after the transition becomes enabled. The transition which samples the minimum firing delay is the one which firing determines the change of marking: the sojourn time in the marking is equal to the minimum sampled delay time of the enabled transitions. The new marking is obtained through the rules of the underlying untemporized PN; the process is started again.

Note: The methodology, which must be adopted for modeling with the proposed TPN, does not allow to disable one enabled transition if another one (concurrently enabled) fires before. The models must verify the so called *persistence* property /74/.

Since the transitions fire in three phases, a transition t_i immediately starts firing provided it is effectively enabled, it sets a timer at the value of its sampled delay instance, and it removes tokens from its input places $\bullet t_i$. Tokens, however, are not deposited into the transition output places $t_i\bullet$ until the transition delay γ_i had elapse, and its timer had reached zero. Tokens are *kept in the transition* during the entire transition delay. Just upon transition firing, the previously removed tokens are deposited into the output places $t_i\bullet$. The token-game of the TPN is graphically described in Fig. 11.

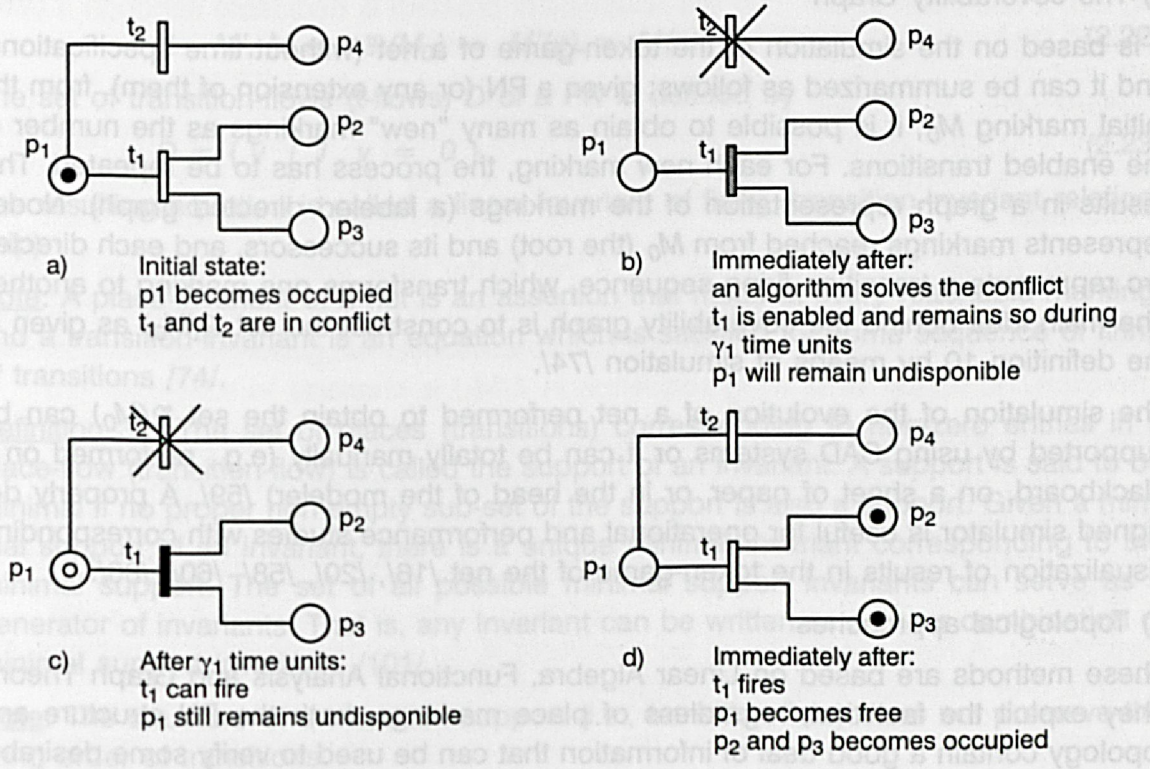


Fig. 11: Token-Game of the TPN

Note: The use of TPNs for modeling a given system allows carrying out a faithful study on the system's behavior. This stage of the analysis is reached by previously qualitatively analyzing the original non-temporized PN. The PN temporizing does not modify the basic structure of a net, therefore maintaining the set of properties of the net /44/. Also, the set of firing sequences of a temporized PN is a subset of the firing sequences

of its non-temporized PN, since the timing information may inhibit certain combinations of firing transitions.

2.5.4 Analysis Methods and Properties of the Nets

A major strength of Petri nets is their support for analysis of many properties and problems associated with the modeled systems /60/, /74/, /101/. Basically, two kinds of analysis can be made: the qualitative and the quantitative analysis. The first one verifies the compliance of certain properties of the net. The quantitative analysis is often called performance evaluation, and it takes into account system specifications, therewith checking the system's compliance with desired performance indexes. A detailed discussion about quantitative methods is beyond the scope of this work. For more details about this kind of analysis, the references /22/, /44/, /69/, /70/, among others, can be consulted.

Qualitative Analysis Methods

Methods of qualitative analysis used in this work are: a) the coverability (reachability) graph method and b) the topological-based approaches.

a) The coverability Graph

It is based on the simulation of the token-game of a net (without time specifications) and it can be summarized as follows: given a PN (or any extension of them), from the initial marking M_0, it is possible to obtain as many "new" markings as the number of the enabled transitions. For each new marking, the process has to be repeated. This results in a graph representation of the markings (a labeled directed graph). Nodes represents markings reached from M_0 (the root) and its successors, and each directed arc represents a transition firing sequence, which transforms one marking to another. The main idea behind the coverability graph is to construct the set $\Re(M_0)$ as given in the definition 10 by means of simulation /74/.

The simulation of the evolution of a net performed to obtain the set $\Re(M_0)$ can be supported by using CAD systems or it can be totally manually (e.g., performed on a blackboard, on a sheet of paper, or in the head of the modeler) /59/. A properly designed simulator is useful for operational and performance studies with corresponding visualization of results in the token-game of the net /16/, /20/, /58/, /60/, /86/.

b) Topological approaches

These methods are based on Linear Algebra, Functional Analysis and Graph Theory. They exploit the fact that, regardless of place markings, both the PN structure and topology contain a good deal of information that can be used to verify some desirable properties of the net.

b1) Study of the state transition equation: The basic idea behind this method is to find a set of equations and inequalities derived from the state transition equation (see definitions 12 and 17) which characterize all reachable markings and some firing sequences of transitions.

Definition 29: Let us consider the equations system

$$x^T \cdot I = 0 \tag{2.16}$$

If $x=v$ is a solution of (Eq. 2.16) and if (Eq. 2.5) is multiplied on the left by v^T, then

$$\forall \quad M', M \in \quad \Re(M_0) \Rightarrow v^T \cdot M' = v^T \cdot M + v^T \cdot I \cdot \Theta \tag{2.17}$$

The set of place-flows (p-flows) E of a PN is defined by

$$E = \{ v \,/\, v^T \cdot I = 0 \} \tag{2.18}$$

The resulting equation

$$v^T \cdot M' = v^T \cdot M \tag{2.19}$$

is called a linear invariant of markings (place-invariant relationship). This says hat the sum of the tokens weighted by v is constant.

Definition 30: Let us consider the equations system

$$I.y = 0 \tag{2.20}$$

If $y=v$ is a solution of (Eq. 2.20), where v is a vector of a positive weighted set of transitions with dimension *cardinality(T)* , the (Eq. 2.5) can be written

$$M'(p) = M(p) + I \cdot v \tag{2.21}$$

then

$$\forall \quad M', M \in \quad \Re(M_0) \Rightarrow M'(p) = M(p) \tag{2.22}$$

The set of transition-flows (t-flows) D of a PN is defined by

$$D = \{ v \;/\; I \cdot v = 0 \} \tag{2.23}$$

The resulting equation is called a linear invariant of firing (transition-invariant relationship).

Note: A place-invariant of a net is an assertion that holds at every reachable marking, and a transition-invariant is an equation which is satisfied for some sequence of firing of transitions /74/.

Definition 31: The set of places (transitions) corresponding to non-zero entries in a place-flow (transition-flow) is called the support of an invariant. A support is said to be minimal if no proper non-empty sub-set of the support is also a support. Given a minimal support of an invariant, there is a unique minimal invariant corresponding to the minimal support. The set of all possible minimal support invariants can serve as a generator of invariants. That is, any invariant can be written as a linear combination of minimal support invariants /101/.

Note: The structure of a transition-support (i.e., transition-flow) does not preserve the firing order of transitions.

Many results on structural analysis of PN and H-L-PN using the information contained in the state transition equation have been already published /60/, /101/. The approach followed here is based on the calculus of place- and transitions-flows of PN and also the symbolic computation of a family of generators of flows in H-L-PN-based models of flexible production systems /18/, /21/, /22/. A detailed discussion about the method is beyond the scope of this work. It is based on mathematical properties of the color

functions of commutative nets, which belong to a ring of commutative diagonalizable endomorphisms and more details about it can be found in /23/, /50/.

b2) Detection of Siphon and Traps: It is a group of structural analysis techniques based on the identification of subsets of places of a net with particular properties /74/, /101/.

Properties of the Nets

By means of the above enumerated analysis methods it is possible to *prove* that a given model has a set of desired properties.

Basically two types of properties can be studied with a Petri net model: a) those depending on the initial marking (*marking-dependent or behavioral properties*), and b) those, independent of the initial marking, called *structural properties*.

Note: Only properties of Petri net models with high significance to flexible production systems and their analysis problems will be here described.

a) Behavioral properties

Reachability: A marking *Mn* is said to be reachable from a marking M_0 if there exists a sequence of firings that transforms M_0 in *Mn* (see definitions 9 and 10). The reachability problem for a Petri net is the problem of finding if $Mn \in \Re(M_0)$ for a given marking *Mn*. It has been shown that the reachability problem is decidable although it takes at least exponential space (and time) to analysis.

Boundedness: A Petri net is said to be *k-bounded* or simply bounded, if the number of tokens in each place does not exceed a finite number *k* for any marking reachable from M_0, i.e., $M(p) \leq k$, $\forall p \in P$, $\forall M \in \Re(M_0)$. A Petri net is said to be *safe*, if it is 1-bounded.

Liveness: A Petri net is said to be *live,* if, no matter what marking has been reached from M_0, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence. This means that a live Petri net guarantees deadlock-free operation, no matter what firing sequence in $\mathcal{L}(M_0)$ is chosen.

Reversibility and Home State: A Petri net is said to be reversible, if M_0 is reachable from *M*, for each marking $M \in \Re(M_0)$. In a reversible net one can always get back to the initial marking. The reversibility condition can be relaxed with the definition of a *home state*. A marking *M'* is said to be a *home state*, if *M'* is reachable from *M*, for each $M \in \Re(M_0)$.

Persistence: A Petri net is said to be persistent, if, for any two enabled transitions, the firing of one transition does not disable the other.

b) Structural properties derived from the analysis of invariant relationships, and siphons and traps.

Controllability: A Petri net is said to be completely controllable, if any marking is reachable from any other marking. This implies that *Rank(I)=m (cardinality of P)(necessary condition).*

Structural boundedness: A Petri net is called structurally bounded, if it is bounded for any finite initial marking M_0.

Conservativity: A Petri net is said to be conservative, if $\forall p \in P, \exists\, v(p) \in \mathbb{Z}$ so that the weighted sum of tokens of Eq. 2.19 is a constant, for every $M \in \Re(M_0)$ and for any fixed initial marking M_0.

Repetitivity: A Petri net is (partially) repetitive, if, and only if there exists a marking M_0 and a firing sequence θ from M_0 such that every (some) transition occurs infinitely often in θ.

Consistency: A Petri net is said to be (partially) consistent, if there exists a marking M_0 and a firing sequence θ from M_0 back to M_0 such that every (some) transition occurs at least once in θ.

2.5.5 High-Level Petri Nets vs. other Tools in Design and Implementation of Discrete-Event Control Systems

The goal of this section is to identify open problems which are to be solved by the design and implementation of DECS and to highlight the advantages of using H-L-PN. Below will be reviewed the main characteristics of the most used tools for modeling and control of the lowest level, i.e., coordination and logic control, of the hierarchical DECS architecture presented in Fig. 4.

One of the main weaknesses of actually implemented DECS is based not on the hardware of the FPS/FPC but on common programming and modelling description techniques for the components of the control structure. These differ in terms of modelling and analysis power, reusability, agility, flexibility and clarity, among others.

In order to classify the most common description techniques and programming languages towards their usability for coordination and logic control purposes, four needs were distinguished by /26/:

- *Sequency*: especially in systems with a large state space it is necessary to describe the *sequence* of states.
- *Concurrency*: for an easier understanding and simplicity of the system's dynamical behavior, concurrent behavior should be taken into account.
- *Efficiency*: given a state of the system, only a few inputs may affect the state and only a few outputs may be changed. No more than the behavior corresponding to these input changes should be specified.
- *Clarity*: have a clear understanding of the input-output behavior of a logic controller, i.e., what is the control applied to the process.

Especially very large and complex DECS that can, for example, be found in FPS have high demands in terms of modelling power and need to be supported methodically in order to keep the models tractable and safe. As a consequence, new needs are now introduced:

- *Complexity*: given a specification with many instances of the same type of submodels, the description method should allow the reduction of complexity by melting these submodels into only one model.

- *Testability*: in order to obtain safe and reliable logic controller models it is absolutely necessary to make sure that certain properties of the model of the logic controller can be guaranteed (e.g., deadlockfreeness). This can be particularly well done by means of formal validation and simulation techniques. Under test are both the correctness of the model itself and the conformity of the model with its specification.
- *Flexibility*: the production systems degree of flexibility highly depends on the integrated control system, i.e., the efficient reconfiguration of controllers. As a consequence, requirements related to the controller software comprise reusability and modularity of logic controller software, as well as portability among different hardware platforms.

In order to fulfill these requirements, is supporting of the demanded needs mandatory for a modeling tool. An overview is now performed on some modelling methods of the components of a DECS at both, coordination and logic control level. Further information on the definition and use of these methods can be found by other authors /5/, /27/, /48/, /57/, /92/, /97/, /102/.

The following two techniques are completely hardware-independent:

- A *state table* contains all the possible internal states, and is therefore only applicable for the accurate analysis of very small systems. According to /26/, none of the first four above listed needs is fulfilled.
- A *state diagram* is more concise than a *state table*. However, it lacks in the representation of concurrency, as it is very hard to identify such parts of the system which are (partially) independent of each other. Compared to a state table, it allows a clear representation of sequential behavior (need 1).

The next four forms of description are more hardware-dependent, and widely used for the specification and programming of code for almost every PLC-based implementation:

- *LD (ladder diagramm)*
  In this model the concurrency can be exploited, but the sequential nature of the behavior remains unclear. Due to the fact, that concurrency is not graphically represented, RLL do not seem to be a suitable tool in terms of modelling logic controllers (need 2 partly fulfilled).
- *Function Chart*
  Contrary to the RLL, symbols are used to represent logic operations such as *AND*, *OR*, and so on. The main advantage of this model is that it is wide-spread and well-accepted among mechanic and electronic specialists as well. Symbols such as flip-flop's allow the representation of sequential processes (needs 1 and 3 fulfilled).
- *Instruction list*
  An instruction list is a sequence of assembler-like statements that allow translating of the logic, represented by the above mentioned Function Chart in a straightforward manner. By means of variables with representative names, it is possible to express the sequential behavior of a process in an adequate manner. Further, it is

possible to use elements that allow structured programming of the logic controller software (needs 1 and 3 fulfilled).

- *Function block diagram*
  Function block diagrams are well-established and a good choice for the control of discrete event systems, if only the classical description techniques and languages are considered. It consists of three basic elements "step", "condition for switching to the next step" and "statement", all of these possess a clear graphical representation and thus allow clear understanding of logic controllers with a low share of concurrency. The main disadvantage is, that concurrent behavior can not be represented (needs 1, 3 and partly 4 fulfilled).

- *Programming languages of the IEC 1131*
  The standard IEC 1131 defines four languages, Instruction List (IL), Structured Text (ST), Ladder Diagram (LD) and Function Block Diagram (FBD). Apart from ST, these programming languages are almost identical with those listed above and fulfill the addressed needs. In addition to this, all of them fulfill need 7, because the portability and reusability of code is supported with the standardization /61/, /85/, /106/ (chapter 7 describes the IEC 1131 more in detail).

The efficient programming of logic controllers demands the fulfillment of all of the above mentioned needs. The next three methods can be classified as *model-based techniques* which are well-known for their capability in modelling concurrent systems.

- *H-L-PN* provide a unified method for design of FPS from hierarchical system descriptions to physical realizations /28/, /102/, /117/ with the following advantages, among others: 1) ease of modeling discrete-event systems characteristics: concurrency, asynchronous and synchronous features, conflicts, mutual exclusion, precedence relationships, non-determinism, and system deadlocks /4/, /11/, /55/, /112/, /118/; 2) excellent visualization of system dependencies; 3) management of local and global information; 4) bottom-up (modular composition) and top-down (stepwise refinement) design methods; 5) ability to generate supervisory control code directly from the graphical H-L-PN representation; 6) ability to check the system for undesirable properties such as deadlock and instability, and to validate code by mathematically-based computer analysis /18/; 7) performance analysis without simulation is possible for many systems, production rates, resource utilization, reliability, and performability can be evaluated /22/, /70/; 8) discrete-event simulation that can be driven from the model /38/; 9) status information that allows model-based real-time monitoring; 10) useful scheduling, because the model contains system precedence relationships as well as constraints on discrete-event performance /39/.

- *Grafcet* is analogous to Petri nets, if places are substituted by "steps" /1/, /25/. The Grafcet structure is an interpreted "logic" graph in which the "etapes" work as flip-flops (in Petri nets the places work as "counters"). According to /26/, needs 1 to 4 are fulfilled. However, most of Petri net analysis techniques (p-invariants, reduction rules, etc.) cannot be applied, if the Grafcet rule "in case of a conflict among transitions, all transitions are firing simultaneously" is used /102/. Due to the straightforward representation of concurrent actions, Grafcet was generally ac-

cepted as an international standard and is now widely spread in industrial applications. Need 7 is not fulfilled as the input/output locations are statically assigned in Grafcet.

- *Colored Petri Nets* (CPN) possess the same structure as Petri nets (need 1, 2 and 6) /59/. The model is extended by features such as colors, functions and guards (see section 2.5.3) in order to allow the reduction of complexity by melting equally-structured sub-models into only one sub-model (need 5).

Although Petri nets and colored Petri nets allow the modelling of coordination and logic controller specifications, they are not intentionally designed for that purpose. In order to specify DECS with Petri nets or colored Petri nets, it is necessary to adapt the behavior of the net in an interpreted manner. Considering this extension, need 3 and 4 are fulfilled for both, modified Petri nets and modified colored Petri nets. These were explained more in detail in the previous sections.

The control structure proposed in this work adapted to a colored Petri net logic control scheme allows the fulfillment of needs 1 to 6. Need 7 is partly fulfilled by generating IEC 1131 code from the proposed CPN-based control structure, as will be described in chapter 7.

Fig. 12 summarizes the evaluation of the above mentioned models and languages. As Petri net and colored Petri net models are not intentionally designed for control purposes, they are highlighted and not rated concerning needs clarity and flexibility.

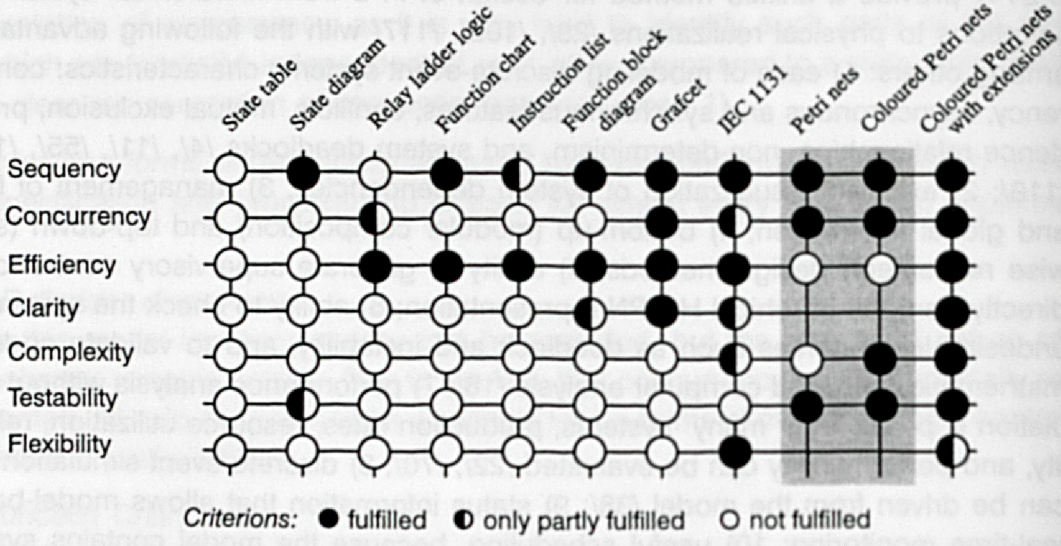


Fig. 12: Degree of Fulfillment according to different Needs for a DECS

2.6 Summary

The main characteristics of flexible production systems and their control systems, i.e., discrete-event control systems, were highlighted, after going through a description and some motivational examples of different production structures.

Most of the widely accepted approaches related to the application of the Petri net theory in the field of production engineering are overviewed and discussed. New sorts of High-Level Petri nets, i.e., parametrized Petri nets and their extensions, were defined and proposed as mathematical-graphical tool to perform the modeling, analysis, validation and implementation of flexible production systems and their control systems. It was shown how recent and on-going research in the field of Petri nets and High-Level Petri nets fits into a new framework proposed in this chapter and issued along this work, which intends to solve many important problems arising in production engineering from the point of view of control and systems theory.

The chapter concludes with an evaluation of design features of existing tools and a comparison of them with the Petri net-based techniques proposed in the following chapters.

3 Formal Specification of Flexible Production Systems using High-Level Petri Nets

Petri nets and extensions of the original definition, e.g., High-Level Petri Nets (H-L-PN), have been proved to be specially adequate for formally specifying parallel and distributed systems such as flexible production systems /21/, /71/, /115/, /117/. Moreover, this kind of graphic-mathematical tool have a well-founded theory of analysis that allows investigating of a great number of modeled system's specifications. The attention of this chapter is pointed to an unique approach based on the theory of H-L-PN able to help solving a big set of design problems with regard to a formal specification of flexible production systems.

Below are overviewed the main concepts and definitions related to H-L-PN and their interpretation when applied to flexible production system's design. Here are presented methods and concepts, which are used to formally specify FPS by means of H-L-PN and to validate system's specifications by means of the analysis of the models. A methodology for the formal specification and validation of functional and performance specifications using qualitative and quantitative analysis of H-L-PN models of FPS is introduced. Finally, a methodology for designing a H-L-PN-based model as skeleton of the FPS coordination control system is proposed and applied to comprehensive examples of FPS.

3.1 High-Level Petri Nets and Flexible Production Systems

It is very important to take into account that the main functions of production systems, whether flexible or not, are to input raw material, to perform a certain number of transformational tasks (i.e., to assemble, disassemble and/or machine parts) and, finally, to output finished parts. Therefore, the designer has to identify the input and output sequences of material parts, and all elementary operations which must be carried out on these parts, and the main characteristics of the resources (i.e., turning, machines, manipulators, AGVs, etc.) involved in each of these operations. In additions to this, all the resources and operations must conform to the following set of constraints [Hardeck 85]:

- The operation are completed in finite time, and they can be decoupled from the point of view of their time specifications.
- One part is submitted to only one operation at a time, and each resource can perform only one task at a time.
- One separate part can be submitted only to transformational or informational functions. A transformational function consists of modifying the physical attributes of the part (shape, constitution, surface, etc.). They are the machining functions: turning, milling, manipulation, assembly, and conditioning functions: chemical treatment, painting, washing, etc.. An informational function consists in verifying that the operations have been accomplished correctly.

The large increase in the number and type of flexible manufacturing and assembly systems (FMS/FAS) being designed and installed has lead to the development and use of a great number of tools which main objective is to reduce the design-time and the costs associated to the implementation phase of the systems. The complexity resulting from the inherent non-linearity, the state space dimension found in most of these systems and problems depending on several characteristics such as flexibility, productivity, agility, costs, required raw materials, human and other resources of the systems, leads to unusual difficulty in design and analysis. With improper design, these systems are prone to deadlock, overflow and degraded performance /117/.

Taken into account the last considerations, methodologies and tools for study and CAD-design are necessary to learn the interactions between different factors for fast and effective evaluation of the qualitative (functional) and quantitative (performance) properties of a set of possible solutions /22/, /67/, /102/.

In the author's opinion, it is very important to perform the modelling, the analysis and the implementation of FPS simultaneously, because none of these processes can be independently adequately developed. In order to achieve the goal, it is interesting to recall that the major advantage of Petri nets is the use of *unique family tools* from the first stage of design until the code generation for the real-time computer-based control of the overall FPS /102/. The static specifications and the dynamic behavior of a complex discrete-event system, like FPS/FPC, may be readily visualized through H-L-PN, which can simulate the system's evolution, analyze its synchronism, describe task concurrency and parallelism, and verify the reachability of a state by means of mathematical analysis and/or simulation. Besides having all the above characteristics as a modelling tool, the H-L-PN model of the systems may be used for its qualitative - functional - and quantitative - performance - analysis, thus becoming an advantageous instrument for validation during the design process of FPS. Indeed, through model validation and verification with H-L-PN, the user of the FPS can rearrange system specifications and objectives due to the easy descriptive characteristics of this graphic-mathematical tool (computer aided engineering (CAE)).

3.2 Modeling Flexible Production Systems with H-L-PN

As stated in chapter 2, section 2.3.1, four kinds of knowledge are mainly necessary for performing the design and modeling of FPS and their DECS with H-L-PN:

(i) Available resources

Resources of a FPC can be described independent on how they are used by a set of physical properties. For example, physical properties of a transport system include its dimensions (e.g., number of places) and transfer mode (e.g., dynamic First-Input-First-Output, static Last-Input-First-Output, etc.). Resources have port-structures to represent "places" at which other resources can be attached, e.g., typical port-structures for a transport system include input-port and output-port. More important is the fact that, constraints can be specified at these port-structures that limit which resources can be attached there. Typically, these constraints would describe properties of the resources

that can be connected at that port-structure or more specifically properties of a port-structure of another resource.

(ii) Functional architecture

A functional architecture specifies a functional decomposition of the FPC and its control system and constraints on their composition. For example, a functional architecture of the controller of a store system has to specify: (a) necessary functions such as store strategy, number of places; (b) constraints on their composition such as, "how the store is accessed by a part", "which addresses are available", "how can parts leave the store"; and (c) constraints on how other functions may be composed with the required functions, e.g., constraints on connecting the store to other components of a production system. For example, the transfer of a pallet from one transport system to the store is an operation which may be generated by the coordination control component as a sequence of functions of local-logic controllers of the transport system and of the store.

(iii) Relation between functions and resources

In general, the mapping between functions, which are to be done in the system, and the resources is many-to-many. A function can be implemented by a set of resources. On the other hand, current resources are often multi-functional (e.g., a robot can load a machine from a transport system, it can assembly parts, it can also unload the machine, etc.).

(iv) Characteristics of a distributed architecture

Lack of global control requires particular means of communication, synchronization, agreement and consensus. Typical examples of these characteristics include the organization of mutual exclusion, distributed termination detection, etc..

Inspired by other reports of the author (see for example /18/, /19/, /21/, this work enlightens new ideas, by considerably improving and generalizing the formalism, necessary to develop a generic model at the job release, flow management, and local control levels of flexible production systems. This formalism is based on the use of a special kind of colored Petri nets tailored for control purposes, and the basic concepts of both, mutual exclusion and structural theories /23/, /50/, /109/, /117/.

3.2.1 Modeling Method

The use of High-level Petri nets (e.g., colored Petri Nets-CPNs) makes possible the creation of compact representations of states, activities, data and events. Nevertheless, a large model can make it difficult to handle the complexity of the modelling, as well as the analysis of its properties /24/. The solution of this problem is considering the system as a set of basic modules from which it can be constructed by composition in a bottom-up manner. A modular approach allows considering different parts of the system independently of one another and also reusing the same module in different systems. The modular modeling method presents also the possibility of composing analysis results of the individual modules, in order to obtain results which are valid for the entire model of the system /21/, /102/.

The method used to develop the CPN models is based on the notion of production- and information flow-preservation and the formal analysis of specifications of the FPS. The specifications of the system can be considered as a list of resources and a list of operations and their desired relationships, together with a description of the way the system is intended to behave /117/.

3.2.2 H-L-PN Model of System Resources

The resources of a FPS include machines, manipulators, buffers, pallets, parts to be processed, and/or products and so on. They can be further divided into two classes: the first one has fixed number of components, e.g., machines, robots, conveyors; the second one has varied number of components, for example, pallets, fixtures, and parts or jobs to be processed /117/.

An operation to be performed in the FPS generally needs both kinds of resources and the completion of it must preserve the flow of parts and information.

This section presents a modular approach for modelling both kinds of resources and also the operations performed by them, using High-Level Petri nets, e.g., Ordered Colored Petri nets (OCPN).

Using Ordered Colored Petri nets (OCPN) as specification tool, the proposed modelling method requires the application of the following basic steps:

- specification of basic color domains and its elements (color tone) (this step maps color tones of a basic color domain to each of the relevant characteristics of the modeled resources, e.g., places of a buffer, sequence of tasks, kinds of processed parts, different types of machines, etc.);
- the universal color domain Ω^* and other complex color domains must be built;
- defining functions $I^+(p,(t, c_t))/I^-(p,(t, c_t))$ associated with the arcs of the net;
- defining Guards associated to some transition of the net;
- synthesis of the net structure.

So, in a first stage and taking into account that:

- transitions and their corresponding firing-modes (occurrence-colors) model operations and events taking place in the real environment, which leads to a change of the system's state, and
- places and their colored markings model states of resources and information about them (herewith can be expressed preconditions for the occurrence of operations and events),

the modeling of a resource of flexible production systems can be summarized as follows:

1) Identify the operations and states of the resource which are required for the production of one item of each manufactured product.

2) Arrange operations, i.e., activities, by the precedence relationships as given in the process plan of a product.

3) For each operation, a transition of the H-L-PN model is drawn, and firing-modes (occurrence-colors) for the transition are defined.

4) In this order: create and label places to represent the states of a resource busy (before and after performing a task, i.e., operation); create and label a place to represent the free-status of that resource (monitor place /9/).

5) Connect each place with an arc from and to the corresponding transitions, according to the precedence relationships as defined above. Corresponding function must be associated to each arc of the net. The function, associated with an arc, models the information- and/or part-flow, e.g., production-flow, throughout the net.

6) Specify the initial marking of the net according to the initial state of the modeled resource. Multiple tokens in a place will indicate a multiplicity of resources or parts, for example, in a buffer-free place, three tokens might represent three positions of a buffer being free at the same time. In the case of H-L-PN models, three tokens, with different attributes, model three positions of a buffer /19/.

The second class of resources (with varied number) plays a special role in the resource-sharing environment /117/. In order to avoid deadlocks and overflows, the appropriate number of initial tokens of the places (initial marking), which model the states of such a resource, has to be carefully defined. For example, the definition of an upper bound for pallets, which has to be loaded into one buffer, can help to avoid deadlocks (some unused space is needed to re-manoeuvre the pallets in case of a temporal deadlock within another resource of the FPS /104/).

When the net is executed, a token in a place (busy state of resource) will indicate that the resource becomes busy. When the net is executed, a token in a place (free state of a resource) will indicate that the resource becomes free. During the net evolution, the marking of the places, resource's *free-state* and *busy-state*, have to be found in mutual exclusion relation.

In the second stage is performed the mapping from each port-structure of the resource to transitions and to occurrence-modes of the OCPN model. This is because, all the ways that a resource can participate in the structure of a flexible production system are those that lead to the functions of the resource, defined by the architecture of the system. These functions are the exchange of information and parts between resources, and are modeled as operations, i.e., transitions or their occurrence-modes.

Below, as an example of application of the above defined 2 stages modeling approach, is presented the behavioral OCPN model of a kind of LIFO storage and a workstation, e.g., machine.

Specifications of a Dynamic LIFO Storage

A LIFO storage receives items (parts, messages), and in the opposed order delivers them (LIFO: Last Input, First Output). Before module, there is a producer system (type of items stored in it). After module, there is a consumer system of these items. No restrictive hypotheses are made regarding the nature of the items transferred into the module under study.

This resource is identified as *dynamic storage* due to its self-motion.

During the storage's loading sequence the items enter it via the last (n-th) position. They will be then transferred from one position to the preceding one. This transfer process goes on, as long as there is a free position in the storage, until the items arrive at the free position nearest to the last loaded position.

During the storage's unloading sequence the items are transferred from the first busy storage position, which is founded into the storage to the next one. This transfer process goes on until the items arrive at the last (n-th) position of the queue. Finally, the items leave the storage from this position, to the exterior, which releases them.

The main characteristic of this module is, that there is a transfer of items between two consecutive positions, during both sequences: loading and unloading of the module. The storage has *n* positions and each of them has capacity for storing a single item "x".

To help depict above considerations Fig. 13 shows a scheme of such a storage.

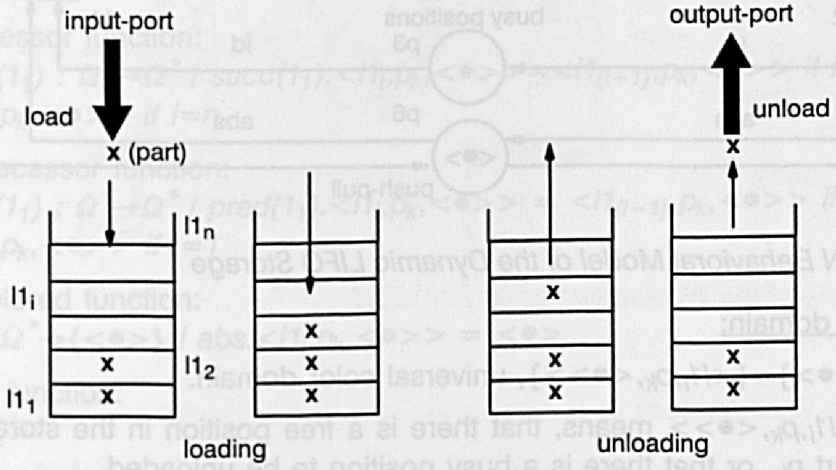


Fig. 13: Dynamic LIFO Storage

Modeling of the Storage

In order to design a CPN model and to verify the above described specifications, it is necessary to consider the loading and unloading operations as a set of places and transitions which model *push-pull* operations. Furthermore, both operations must be performed in mutual exclusion relation. The above points lead to the OCPN model of Fig. 14, which is a faithful reflection of the preceding considerations.

The basic color domains associated with the firing of transitions and with the marking of the places, the transitions, the places, the color functions of the net, the guards, and the initial marking of places are described below.

Basic color domains:

L1 (set of storage positions)$=\{<l1_i>,\ 1 \le i \le n\}$ basic color domain, where $<l1_i>$ is the position *i* in the storage

$P = \{p_k,\ k \in [1:m]\}$ set of parts to be transported on the LIFO

{<•>} basic color domain, where <•> is the uncolored token /74/.

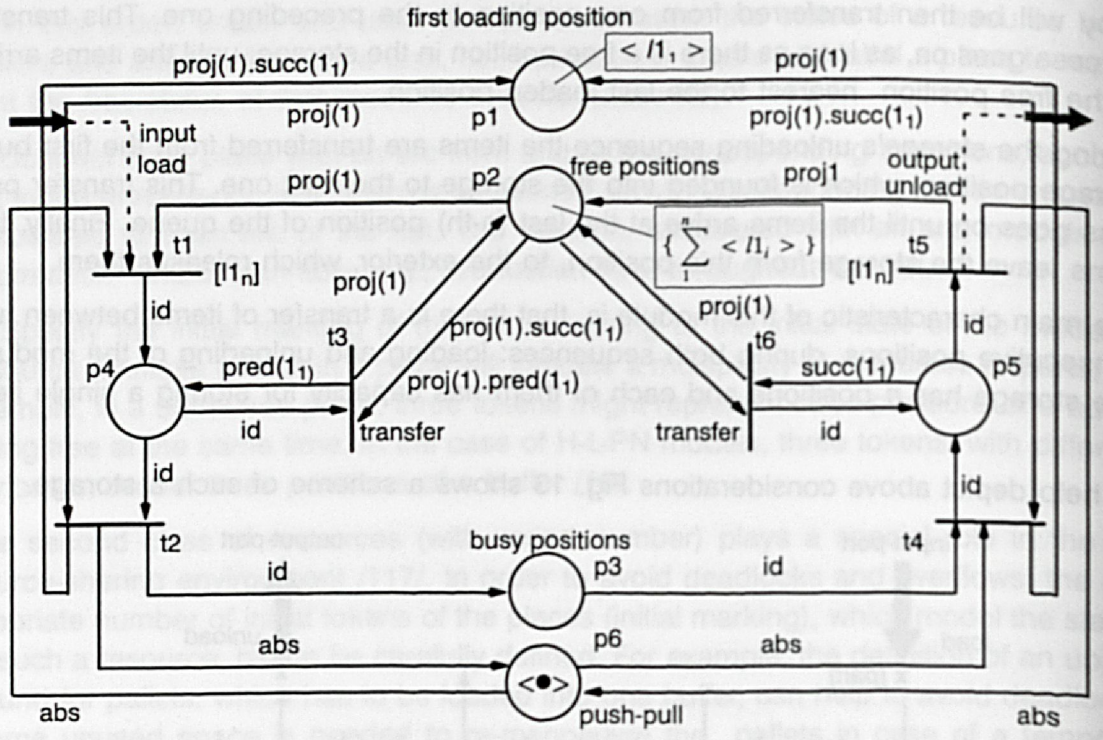


Fig. 14: OCPN Behavioral Model of the Dynamic LIFO Storage

Universal color domain:

$\Omega^* = L1 \times P \times \{<\bullet>\} = \{<l1_i, p_k, <\bullet>>\}$, universal color domain.

The element $<l1_i, p_k, <\bullet>>$ means, that there is a free position in the storage to be loaded by a part p_k, or that there is a busy position to be unloaded.

Transitions:

$T = \{t1, t2, t3, t4, t5, t6\}$

t1: models the operation "start of a push operation"

t2: models the operation "end of a push operation"

t3: models the operation "transfer an item p_k from the storage position, $<l1_i>$, to the preceding position, $<l1_{(i-1)}>$, during the storage's loading sequence"

t4: models the operation "start of a pull operation"

t5: models the operation "end of a pull operation"

t6: models the operation "transfer an item p_k from the storage position, $<l1_i>$, to the next position, $<l1_{(i+1)}>$, during the storage's unloading sequence".

Places:

$P = \{p1, p2, p3, p4, p5, p6\}$

p1: if marked means the position nearest to the last loaded storage position is free.

p2: if marked means the positions in the storage are free.

p3: if marked means the positions in the storage are busy.

p4: if marked means the positions during the storage's loading sequence are transitory busy.

p5: if marked means the positions during the storage's unloading sequence are transitory busy.

p6: if uncolored marked means whether push nor pull operations are being performed. It is also called *mutex*.

Standard color functions:

- Projection function:
  $proj(1): \Omega^* \rightarrow L1 \;/\; proj(1).<l1_i,p_k,<\bullet>>=<l1_i>$
- Identity function:
  $id : \Omega^* \rightarrow \Omega^* \;/\; id.<l1_i,p_k,<\bullet>>=<l1_i,p_k,<\bullet>>$
- Successor function:
  $succ(1_1) : \Omega^* \rightarrow \Omega^* \;/\; succ(1_1).<l1_i,p_k,<\bullet>>=<l1_{(i+1)},p_k,<\bullet>>$ *if* $i<n$,
  $<l1_1,p_k,<\bullet>>$ *if* $i=n$
- Predecessor function:
  $pred(1_1) : \Omega^* \rightarrow \Omega^* \;/\; pred(1_1).<l1_i,p_k,<\bullet>> = <l1_{(i-1)},p_k,<\bullet>>$ *if* $i>1$,
  $<l1_n,p_k,<\bullet>>$ *if* $i=1$
- Decolored function:
  $abs: \Omega^* \rightarrow \{<\bullet>\} \;/\; abs.<l1_i,p_k,<\bullet>> = <\bullet>$
- Ring function:
  $R<l1_i>$

Initial marking of places:

- *p1* is initialized with value $m_0(p1)=<l1_1>$, because this work proposes to begin the storage loading process onto this specific position.
- *p2* is initialized with value $m_0(p2)=\Sigma_{1 \le i \le n}<l1_i>$, because the storage has *n* free positions.
- *p3* has initial marking *0*, that means, it is unmarked, because the storage is empty at the beginning of its work.
- *p6* is initialized with the marking $m_0(p6)=<\bullet>$, because the function of the mutex.

All the other places of the CPN have initial marking *0*, that is, they are unmarked, before the *push-pull* operations are performed.

Guards:

The closer look at transitions *t1* and *t5* will show that they have associated a guard $[l1_n]$. This tells that only tokens representing $<l1_n>$ can move from place *p2* to place *p4* or from place *p5* to place *p2* (because the guard for all colors $\Sigma_{(1 \le i \le n-1)}<l1_i>$ evaluates to false and thus prevents enabling).

Specifications of a Workstation

The robot can be basically found within three possible states: free, busy before it works and busy after it worked. There are three major tasks to be performed that are related with the robot's operation: loading and unloading the work position with a part, and part is processed by the robot. Once these operations are started, they cannot be interrupted until they finish. Up the work position of the robot, there is a producer system (type of items to be processed in it). Down the work position of the robot, there is a consumer system of these items. No restrictive hypothesis are made regarding the nature of these items. Another robot or a human operator is responsible for loading and unloading the work position (see Fig. 15).

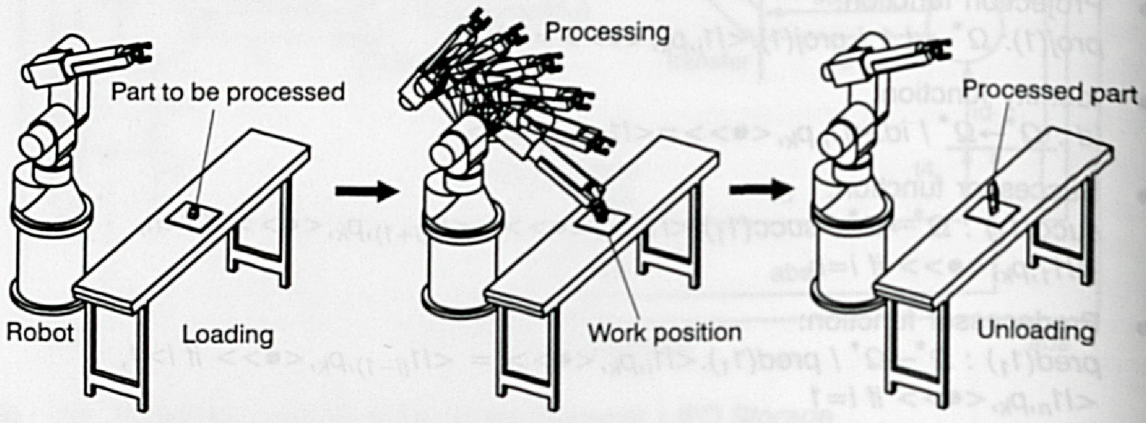


Fig. 15: Schema of the Workstation to be Modeled

Modeling of the Workstation

The above named behavioral specifications lead to the OCPN model of Fig. 16, which is a faithful reflection of the preceding considerations.

Below are described the basic color domains associated with the firing of transitions and the marking of the places, the transitions, the places, the color functions of the net, the guards, and the initial marking of places.

Basic color domains:

$M = \{m_i,\ i \in [1:1]\}$ identification of the robot type

$P = \{p_k,\ k \in [1:m]\}$ set of parts processed on the work position of the robot

$\{<\bullet>\}$ basic color domain, where $<\bullet>$ is the uncolored token /74/.

Universal color domain:

$\Omega^* = M \times P \times \{<\bullet>\} = \{<m_i, p_k, <\bullet>>\}$, universal color domain.

The element $<m_i, p_k, <\bullet>>$ means, that there is a part $<p_k>$ to be loaded on the work position of the robot $<m_i>$, processed in it, and/or unloaded from this position to the exterior.

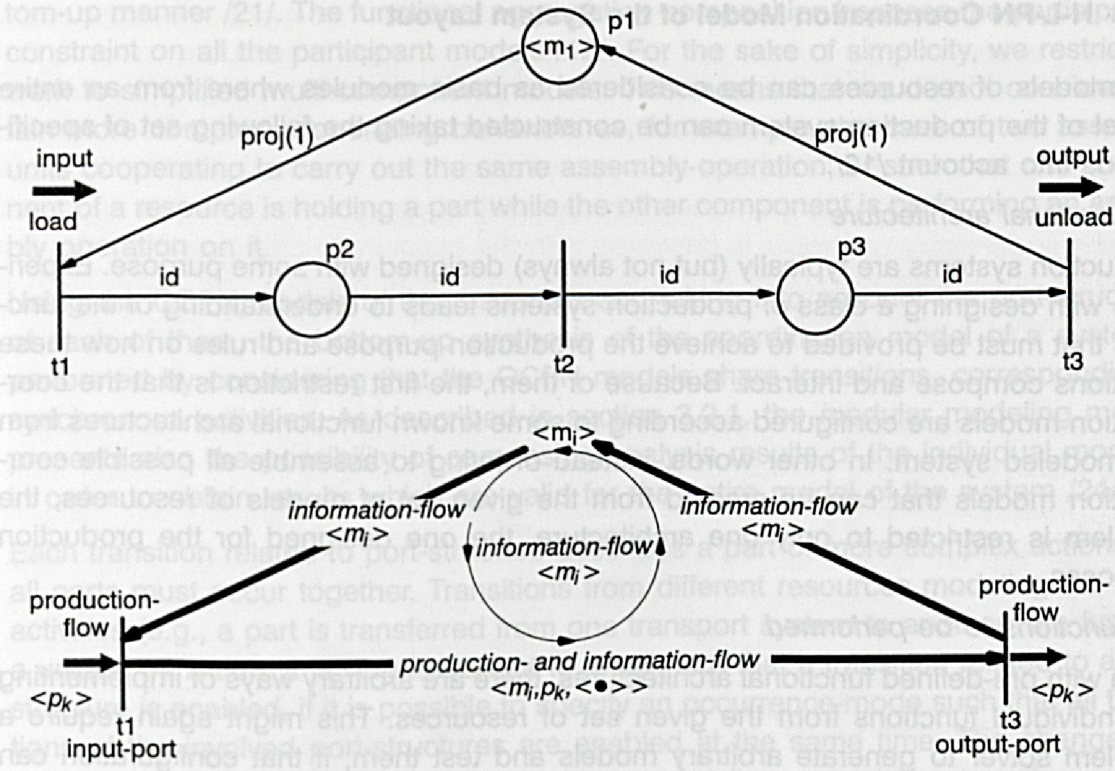


Fig. 16: OCPN Behavioral Model of a Machine

Transitions:

T={t1, t2, t3}

t1: models the operation "loading the work position of the robot".

t2: models the operation "robot processes a part".

t3: models the operation "unloading the work position of the robot".

Places:

P={p1, p2, p3}

p1: if marked models the state "robot free"

p2: if marked models "work position of the robot busy before it processes a part"

p3: if marked models "work position of the robot busy after it processed a part"

Standard color functions:

Projection function: $proj(1): \Omega^* \rightarrow M \;/\; proj(1).<m_i,p_k,<\bullet>>=<m_i>$

Identity function: $id : \Omega^* \rightarrow \Omega^* \;/\; id.<m_i,p_k,<\bullet>>=<m_i,p_k,<\bullet>>$

Initial marking of places:

p1 is initialized with value $m_0(p1)=<m_1>$

p2 and *p3* are initialized with value *0*, that is, they are unmarked.

3.2.3 H-L-PN Coordination Model of the System Layout

The models of resources can be considered as basic modules where from an entire model of the production system can be constructed taking the following set of specifications into account /12/:

(i) Functional architecture

Production systems are typically (but not always) designed with some purpose. Experience with designing a class of production systems leads to understanding of the functions that must be provided to achieve the production purpose and rules on how these functions compose and interact. Because of them, the first restriction is that the coordination models are configured according to some known functional architectures from the modeled system. In other words, instead of trying to assemble all possible coordination models that can be created from the given set of models of resources, the problem is restricted to only one architecture, the one designed for the production purposes.

(ii) Functions to be performed

Even with pre-defined functional architectures, there are arbitrary ways of implementing the individual functions from the given set of resources. This might again require a problem solver to generate arbitrary models and test them, if that configuration can indeed provide the desired functions. However, it is important to repeat here that in many systems is possible to identify some particular configuration of modeled resources that are crucial to implementing some functions in the whole coordination model.

(iii) Connectivity between resources

After identifying the port-structures of the resources and a set of structural composition constraints derived from the structure of the system, a simple way to develop a coordination model is with a bottom-up synthesis-and-validation method. In this case, a modeler starts with one of the available models of resources and creates a viable coordination model by composing other resources satisfying connectivity constraints and taking into account the functions which are to be performed in the system (modeling form resources to functions). The resources share port-structures and they exchange information and parts to be processed in the system, e.g., pallets with processed or assembled parts, by means of the ports. The main physical constraint for a correct functioning of the system is the preservation of information and parts in movement, and this constraint has to be considered for each step during the synthesis of the coordination model.

At this point, in addition to an ordering of manufacturing functions, an engineer will have established a set of resources capable of providing these functions together with their validated OCPN models.

The layout model of the system, i.e., coordination model of resources, is then obtained by taking into account both, competence and cooperation, relationships between resources, constraints of the ordering of functions above cited by selecting abstract routes for the workpieces, and by composing the models of the resources in a bot-

tom-up manner /21/. The functional aggregation composition imposes the participation, constraint on all the participant models /12/. For the sake of simplicity, we restrict our work to simplified multi-component models. This means that we do not consider certain more complex cooperating behaviors as, for example, the case of two assembly units cooperating to carry out the same assembly-operation, or such that one component of a resource is holding a part while the other component is performing an assembly operation on it.

Using the OCPN models of the resources and taking into account the port-structures of each of them, the bottom-up synthesis of the coordination model of a system is performed by considering that the OCPN models share transitions, corresponding to synchronous activities. As described in section 3.2.1, the modular modeling method presents also the possibility of composing analysis results of the individual modules, in order to obtain results which are valid for the entire model of the system /24/.

Each transition related to port-structure describes a part of more complex actions and all parts must occur together. Transitions from different resources modeling the same activities (e.g., a part is transferred from one transport system to another) are fused in a unique transition. It is very important to recall here that a transition related to a port-structure is enabled, if it is possible to specify an occurrence-mode such that all transitions of the involved port-structures are enabled at the same time. The change produced by the occurrence of the transitions modeling a port-structure for a given occurrence-mode is the sum of changes produced in each involved OCPN model of resources. A guard may be shared by all transitions belonging to a port-structure, and will be bound to the same value for all of these.

To help ground the modeling methodology, a simple flexible production cell example is presented as follows.

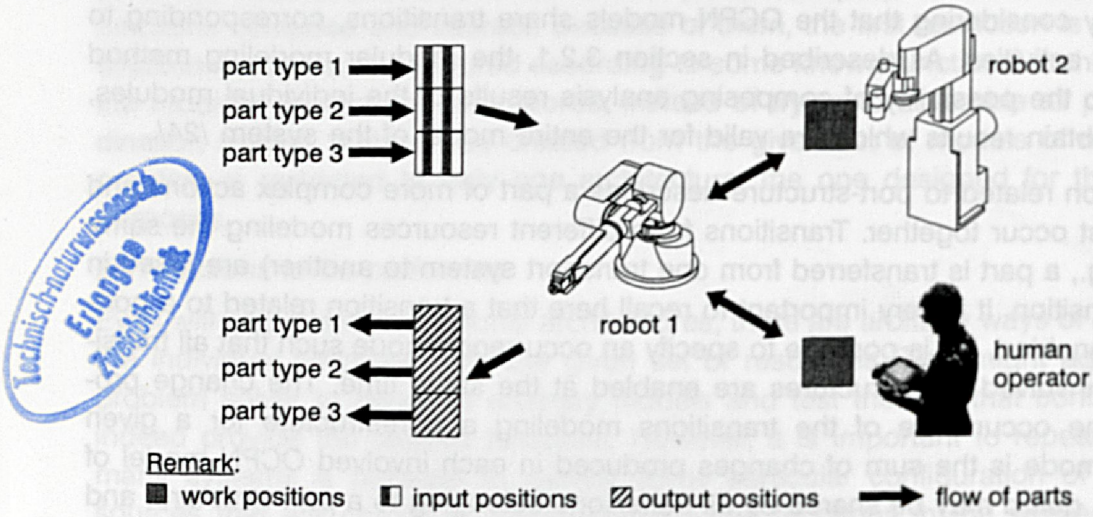
Specifications of a Flexible Production Cell

The system consists of two different robots, e.g., robots 1 and 2, and a human operator, and is sketched in Fig. 17. There are three kinds of parts to be processed in the cell, e.g., part types 1, 2 and 3. The parts of type 1 from the input position must be processed by the robot 2. The parts of types 2 and 3 from the corresponding input position must be processed by the human operator. The robot 1 is used for loading the work positions of the robot 2 and of the human operator from an input position, and for unloading both resources to an output position. Each of the above presented resources has capacity for handling only one part at time. Once a resource starts working on a part, it can not be interrupted until the work is completed. The robot 1 is shared by the human operator and also the robot 2, it can either serve the human operator or robot 2, but not both simultaneously.

The following characteristics are embedded in this system: it is event-driven, asynchronous, and sequential; it exhibits concurrency, conflict, mutual exclusion and non-determinism.

Such a simple production cell can contain a system deadlock which may result improperly triggering a sequence of events. Suppose that, the robot 2 is still processing a part

of type 1 and the human operator finished his task. While this is occurring, the robo 1 grasps a part of type 1 from the input position and tries to load the robot 2, the system is deadlocked. This is because the work position of robot 2 is already full and it cannot be loaded. The work position of the human operator cannot be unloaded either, because the robot 1 is busy by a part of type 1 and it is not available. To avoid such a catastrophic failure, one can adopt one of two approaches. The first approach is that a supervisory controller is designed with the capacity to detect and resolve the deadlock during the operation of the system /117/. It is obvious that using such a method may be very costly.



Technisch-naturwissensch. Zweigbibliothek Erlangen

Fig. 17: Layout Specifications of a Sample Flexible Production Cell

The second approach, here proposed, is to design a model of the cell with desirable properties, in this case, freedom from deadlock. In the example, the model has to be designed so that the robot 1 (a shared resource) grasps a part to be loaded on the work position of the human operator or the robot 2 if, and only if, the corresponding position is free.

Modeling of the Cell

In order to design a H-L-PN model of the system layout with desirable behavioral properties, there arise the concepts of *logic and physical operations*.

Definition 32: *Logic operation* is related with the *information handling* in the model and it has no meaning from the point of view of the processes developed in the modeled system.

Definition 33: *Physical operation* is related to the *processes developed* in the system, e.g., machining, assembly, grasping, etc.

Both kinds of operations are modeled by means of transitions of H-L-PN.

By considering the functional architecture of the whole system and the functions to be performed by the resources of the cell, as previously discussed, the modeling of the system with the proposed OCPN starts with the definition of the basic and universal color domains.

Basic color domains:

$M = \{m_1(robot\ 2), m_2(human\ operator)\}$ identification of the work position. m_1: *robot 2;* m_2: *human operator*

$R = \{r_j: r_1(robot\ 1)\}$

$P = \{p_k,\ k \in [1:3]\}$ set of parts to be processed on the machine

$\{<\bullet>\}$ basic color domain, where $<\bullet>$ is the uncolored token /74/.

Universal color domain:

$\Omega^* = M \times R \times P \times \{<\bullet>\} = \{<m_i, r_j, p_k, <\bullet>>\}$, universal color domain.

The element $<m_i, r_j, p_k, <\bullet>>$ means, that there is a part $<p_k>$ to be loaded on the work position $<m_i>$, processed in it, and/or unloaded from the work position to the exterior. The loading and unloading operations are performed by the robot r_1.

After having defined the basic color domains for each resource of the cell, the corresponding functions, components (e.g., places and transitions) and guards of the OCPN model have to be also defined.

Standard color functions:

Projection functions

$proj(1): \Omega^* \rightarrow M \ / \ proj(1).<m_i, r_j, p_k, <\bullet>> = <m_i>;$

$proj(2): \Omega^* \rightarrow R \ / \ proj(2).<m_i, r_j, p_k, <\bullet>> = <r_j>$

$proj(13): \Omega^* \rightarrow P \ / \ proj(13).<m_i, r_j, p_k, <\bullet>> = <m_i, p_k>$

Identity function

$id: \Omega^* \rightarrow \Omega^* \ / \ id.<m_i, r_j, p_k, <\bullet>> = <m_i, r_j, p_k, <\bullet>>;$

Transitions:

$T = \{t1, t2, t3, t4, t5, t6, t7\}$

t1: models the logic condition "r_1 grasps a part to be loaded in one of the both work positions from the input position". The target work position is reserved for such operation.

t2: models the physical operation "r_1 grasps a part from an input position and loads it in the target work position".

t3: models the logic operation "r_1 achieves free status".

t4: models the physical operation "part loaded on target work position is processed".

t5: models the logic operation "r_1 grasps a part from a work position and unloads it to the corresponding output position". The work position maintains its "reserved condition".

t5: models the physical operation "r_1 grasps a part from a work position and unloads it to the corresponding output position".

t6: models the logic operation "r_1 and the work position achieves free status".

Places:

$P=\{p1, p2, p3, p4, p5, p6, p7, p8, p9, p10\}$

p1: if marked shows the state "human operator, robot 2 free".

p2: if marked means "robot 1 free".

p3: if marked means "robot 1 can grasp a part and load a work position".

p4: if marked means "robot 1 loaded a work position".

p5: if marked means "human operator or robot 2 can process a part".

p6: if marked means "human operator or robot 2 finished".

p7: if marked means "robot 1 can grasp a part and unload a work position".

p8: if marked means "robot 1 unloaded a work position".

p9: if marked means "parts to be grasped by the robot 1 and loaded into one work position".

p10: if marked means "processed parts".

An adequate initial marking of the net has to be defined now.

p1 is initialized with value $m_0(p1)=<m_1+m_2>$

p2 is initialized with value $m_0(p2)=<r_1>$

p3, p4,...., p10 are initialized with value *0*, that is, they are unmarked.

Guards:

In order to fulfill the specifications of the work-plan associated to each part, the following guards have to be associated to some transitions of the net.

$G\&_1(t1)=G\&_1(t5)=(m_1 \wedge p_1), G\&_2(t1)=G\&_2(t5)=(m_2 \wedge p_2), G\&_3(t1)=G\&_3(t5)=(m_2 \wedge p_3)$

$G(t1)=G\&_1 \vee G\&_2 \vee G\&_3;\ G(t5)=G\&_1 \vee G\&_2 \vee G\&_3$

Finally, taking into account the connectivity between resources, the OCPN model of the cell can be generated (see Fig. 18). The transition from the functional description and behavioral specifications to the OCPN model is accomplished by identifying several special arrangements among functions which are to be performed in the cell, and instantiating corresponding OCPN elements into the whole OCPN model under construction /12/. This approach needs an overall synchronization of the subnets, i.e., models of resources, by using, for example, a kind of *rendez-vouz* mechanism /25/.

Comparison between the OCPN models of Fig. 16 and 18 allows highlighting the significance of modeling, with the proposed modular approach. Topological analysis of the structure of the OCPN in Fig. 18 shows the existence of three sub-OCPNs like this one of Fig. 16, that is, models of the human operator and the robot 2 fused into one structure, and twice the model of the robot 1, respectively.

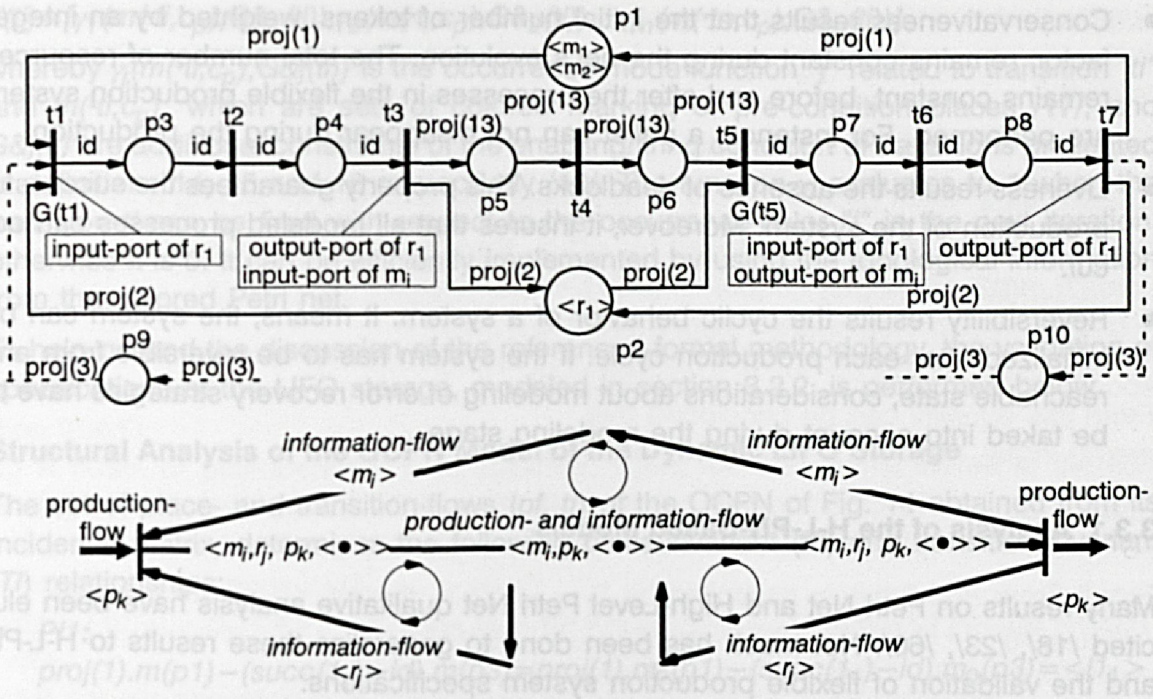


Fig. 18: OCPN Model of the Sample Flexible Production Cell of Fig. 17

3.3 High-Level Petri Net Properties and Specifications of the Systems

Design process of flexible production systems is quite a complex task. Thus, it is very important to validate the significant steps of model building/transformation before going on to the implementation /102/. Validation analysis aims at verifying the correctness of system design and at verifying all initial specifications. This kind of analysis is achieved through description, namely, the model of the system (in this work: H-L-PN).

Faced with the growing complexity of the flexible production systems, H-L-PN represent an interesting compromise of power and analytical capacity on the already built models /45/. One of the major advantages of Petri nets is the ability to analyze them for properties related to the specifications of the modeled systems. Petri nets have a well-founded mathematical theory, which allows validation of the specifications of the modeled systems by means of qualitative and quantitative analysis of the net properties before the implementation is performed.

From the set of properties addressed in chapter 2, section 2.4.4, boundedness or safeness, conservativeness, liveness, and reversibility are of high relevance for the qualitative validation of system's specifications. Their significance to flexible production systems is stated as follows /117/:

- Boundedness or safeness results the absence of capacity overflows. Safeness is the special case of 1-bounded. For instance, all resources of a given flexible production system have finite capacity. Safeness of a model indicates the availability of only a single modeled resource.

- Conservativeness results that the initial number of tokens, weighted by an integer factor, remains constant during the net's evolution. The total number of resources remains constant, before and after the processes in the flexible production system are performed. For instance, a pallet can not disappear during the production.
- Liveness results the absence of deadlocks. This property guarantees the successful production of the system. Moreover, it insures that all modeled processes can occur.
- Reversibility results the cyclic behavior of a system. It means, the system can be initialized after each production cycle. If the system has to be reversible from any reachable state, considerations about modeling of error recovery strategies have to be taked into account during the modeling stage.

3.3.1 Analysis of the H-L-PN-Based Models

Many results on Petri Net and High-Level Petri Net qualitative analysis have been elucited /18/, /23/, /60/. Not much has been done to generalize these results to H-L-PN and the validation of flexible production system specifications.

The kind of H-L-PN used in this work as modeling tool is a sub-class of colored nets which color functions belong to a ring of commutative diagonalizable endomorphism /23/. Mathematical properties of these color functions allow a symbolic computation of family of flows, by means of a generalization of the Gauss elimination method applied to the incidence matrix of the nets and its transpose.

Based on a method proposed in /23/ for calculating linear invariants deduced from the left annihilators (flows) of a H-L-PN's incidence matrix, this work proposes an approach for validating flexible production system's specifications, using the information contained in the calculated linear invariant structures /21/.

3.3.2 Validation of Specifications of Resources

The information contained in the structures of transition- and place-flows and the corresponding transition- and place-invariant relationships of a H-L-PN model allows performing the validation of many logical and technological properties of the resources of flexible production systems, without resorting to simulate the net's evolution (token-game).

Taking into account the definitions 14 and 17 in chapter 2, the behavior of a colored Petri net is determined by the net's structure. It can also be expressed by means of occurrence-mode function defined below.

Remark: The definition of the occurrence-mode function, used in this work, is based on the definitions presented in /4/.

Definition 34: The occurrence-mode function is defined for each transition of a net as Γ: $[R(M(p))\times G(t)] \rightarrow \{0,1\}$ and it can be seen as a functional vector of Boolean functions:

$\Gamma(ti)=[\gamma_1(m(\bullet ti,c_p),G\&_1(ti)),\gamma_2(m(\bullet t_i,c_p),G\&_2(ti)),...,\gamma_n(m(\bullet tn,c_p),G\&_n(ti))]$

whereby $\gamma_j(m(\bullet ti,c_p),G\&j(ti))$ is the occurrence-mode function *"j"* related to transition *"ti"* and $m(\bullet ti,c_p)$, which are sets of colored marking of pre-condition places /17/, and $G\&j(ti)$ are additional constraints of the enabling/firing condition of transitions presented in definitions 14, 15 and 16 respectively /40/. The function γ_j evaluates to *1* when the transition *ti* can be fired with respect to the occurrence-color *"j"* in the next iteration, otherwise it is *0*. It can be efficiently implemented by using the topological information from the colored Petri net.

To help ground the discussion of the referenced formal methodology, the validation of specifications of the LIFO storage, modeled in section 3.2.2, is performed below.

Structural Analysis of the OCPN Model of the Dynamic LIFO Storage

The set of place- and transition-flows (*pf, tf*) of the OCPN of Fig. 14 obtained from its incidence matrix, determines the following place-invariant (*PI*) and transition-invariant (*TI*) relationships:

- *PI1:*
  $proj(1).m(p1)-(succ(1_1)-id).m(p3)=proj(1).m_0(p1)-(succ(1_1)-id).m_0(p3)=<1_1>$
- *PI2:* $proj(1).m(p2)+id.[m(p3)+m(p4)+m(p5)]=$
  $=proj(1).m_0(p2)+id.[m_0(p3)+m_0(p4)+m_0(p5)]=<1_1>+<1_2>+.....+<1_n>=$
  $=\Sigma_{1\le i\le n}<1_i>$
- *PI3:* $proj(1).R<1_i>.m(p1)=<1_1>+<1_2>+...+<1_i>+...+<1_n>=\Sigma_{1\le i\le n}<1_i>$
- *PI4:* $id.[m(p4)+m(p5)]+abs.m(p6)=id.[m_0(p4)+m_0(p5)]+abs.m_0(p6)=<\bullet>$
- *tf1:* $[<1_n,p_k,<\bullet>>\ id\ R<1_i>\ id\ <1_n,p_k,<\bullet>>\ R<1_i>]^T$
- *TI1:* $I(p,(t,c_t)).tf1=I(p,(t,c_t)).[<1_n,p_k,<\bullet>>\ idR<1_i>id<1_n,p_k,<\bullet>>\ R<1_i>]^T=0$
- *TI2:* $I(p,(t,c_t)).tf2=I(p,(t,c_t)).[0\ 0\ R<1_i>\ 0\ 0\ 0]^T=0$
- *TI3:* $I(p,(t,c_t)).tf3=I(p,(t,c_t)).[0\ 0\ 0\ 0\ 0\ R<1_i>]^T=0$

Validation of the Specifications of the LIFO Storage by Means of the Analysis of Invariant Relationships

Axiom 4: Actions in the LIFO system are associated to firing-modes of transitions of the OCPN model.

Axiom 5: Action in the LIFO system can be performed only if the corresponding occurrence-mode function has the value true, i.e., 1, and the enabled firing-mode is fired.

Proposition 1: Place *p1* has capacity for a single color marking, the cardinality of its marking is one, which initial value corresponds to the storage position to be loaded at the end of the first push operation.

Proof: If the initial marking of *p1* is $m_0(p1)=<1_1>$, and is from *PI3*, if this is multiplied by $<1_i>$, then $\forall<1_i,p_k,<\bullet>>\in\Omega^*$

$proj(1).R<1_i>.m(p1).<1_i>=(<1_1>+<1_2>+...+<1_i>+...+<1_n>).<1_i>=1$

From this equality it is possible to conclude that $\exists\ <1_i>\ /\ m(p1)=<1_i>$ and $card[m(p1)]=\#[m(p1)]=1$, which proves the proposition true.□

Lemma 1: At any time, the marking of place *p1* is the color $<l1_i>$, which corresponds to the storage position, to be loaded, immediately following the sequence of push operations.

Proof: Immediate from the structure of place-invariant relationships *PI1* and *PI3*.□

Proposition 2: Each position of the LIFO storage has capacity for storing a single item.

Proof: From *PI2*, if this is multiplied by $<l1_i>$, then $\forall <l1_i,p_k,<\bullet>> \in \Omega^*$

$\{proj(1).m(p2)+id.[m(p3)+m(p4)+m(p5)]\}.<l1_i>=$

$=\{proj(1).m_0(p2)+id.[m_0(p3)+m_0(p4)+m_0(p5)]\}.<l1_i>=$

$=(<l1_1>+<l1_2>+...+<l1_n>).<l1_i>=(\Sigma_{1\le i\le n}<l1_i>).<l1_i>=1$

From this equality can be concluded that

$proj(1).m(p2).<l1_i> \le 1$ which proves the proposition true.□

Proposition 3: The LIFO storage has *n* positions and the maximum number of items to be stored simultaneously in this system is *n* (LIFO storage capacity).

Proof: From *PI2*, if this is multiplied by $\Sigma_{1\le i\le n}<l1_i>$, then $\forall <l1_i,p_k,<\bullet>> \in \Omega^*$

$\{proj(1).m(p2)+id.[m(p3)+m(p4)+m(p5)]\}.\Sigma_{1\le i\le n}<l1_i>=$

$=\{proj(1).m_0(p2)+id.[m_0(p3)+m_0(p4)+m_0(p5)]\}.\Sigma_{1\le i\le n}<l1_i>=$

$=(<l1_1>+<l1_2>+...+<l1_n>).(\Sigma_{1\le i\le n}<l1_i>)=(\Sigma_{1\le i\le n}<l1_i>).(\Sigma_{1\le i\le n}<l1_i>)=n$

From above equality can be concluded that

$proj(1).m(p2).\Sigma_{1\le i\le n}<l1_i> \le n$ which proves the proposition true.□

Proposition 4: At any time, all the positions of the LIFO storage can be found at one of the following states: free or busy.

Proof: If *PI2* is multiplied by $<l1_i,p_k,<\bullet>>$, then $\forall <l1_i,p_k,<\bullet>> \in \Omega^*$

$\{proj(1).m(p2)+id.[m(p3)+m(p4)+m(p5)]\}.<l1_i,p_k,<\bullet>>=$

$=\{proj(1).m_0(p2)+id.[m_0(p3)+m_0(p4)+m_0(p5)]\}.<l1_i,p_k,<\bullet>>=$

$=(<l1_1>+<l1_2>+...+<l1_n>).<l1_i>=(\Sigma_{1\le i\le n}<l1_i>).<l1_i>=1$

According to this, main conclusion may be derived

$proj(1).m(p2).<l1_i,p_k,<\bullet>>+id.m(p3).<l1_i,p_k,<\bullet>>+id.m(p4).<l1_i,p_k,<\bullet>>+$

$+id.m(p5).<l1_i,p_k,<\bullet>>=1$

and it follows, that the places *p2, p3, p4* and *p5* are in mutual exclusion in regard to the color $<l1_i,p_k,<\bullet>>$.

Since the places *p2* and *p3* model busy and free storage positions respectively, the mutual exclusion condition between them proves partially the proposition true. The places *p4* and *p5* model transitory busy storage positions during the push and pull operation respectively, so both markings are in mutual exclusion in regard to the same color $<l1_i,p_k,<\bullet>>$, which proves completely the proposition true.□

Proposition 5: The operations push and pull are in mutual exclusion.

Proof: From *PI4*, if this is multiplied by $<\bullet>$, then $\forall <l1_i,p_k,<\bullet>> \in \Omega^*$

$\{[abs.m(p4)+abs.m(p5)]+m(p6)\}.<\bullet>=<\bullet>.<\bullet>=1$

From above equality can be concluded that

$[abs.m(p4)+abs.m(p5)].<\bullet> \le 1$ which proves the proposition true.□

Proposition 6: In the proposed LIFO storage model are the items sequentially stored.

Proof: It will be considered that two states of the storage module are feasible, that is:

1) It is supposed that only one item is stored in the storage, and it is occupying the position $<I1_j>$, then $\forall <I1_j,p_k,<\bullet>> \in \Omega^*$
$m(p3).<I1_j,p_k,<\bullet>>=1$
Taken into account the place-invariant relationship *PI1*, and the initial marking $m_0(p1)=<I1_1>$, then
$(-id+succ(1_1)).m(p3)=-<I1_j,p_k,<\bullet>>+<I1_{(j+1)},p_k,<\bullet>>$ and
$m(p1)=<I1_{(j+1)}>-<I1_j>+<I1_1>$.

From last equalities and according to proposition 1, the place p1 has cardinality one, main conclusions are $<I1_j>=<I1_1>$ and $m(p1).<I1_2>=1$ which proves the proposition true.□

2) A push- or pull-operation is in advance. There are $k \geq 0$ items stored in the LIFO storage following the position $<I1_j>$, that is, the initial marking of p1 was $m0(p1)=<I1_{(j+1)}>$. There is another item stored in the position $<I1_r>$ of the storage, then $\forall <I1_j,p_k,<\bullet>> \in \Omega^*$
$m(p3).(<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+2)},p_k,<\bullet>>+...+<I1_{(j+k)},p_k,<\bullet>>+<I1_r,p_k,<\bullet>)=$
$=k+1$

a) If a push operation is in advance, from place-invariant relationship *PI4*
$m(p4).<\bullet>=1;\ m(p5)=0;\ m(p6)=0$

b) If a pull operation is in advance, from place-invariant relationship *PI4*
$m(p5).<\bullet>=1;\ m(p4)=0;\ m(p6)=0$

Then $\forall <I1_j,p_k,<\bullet>> \in \Omega^*$
$(-id+succ(1_1)).m(p3)=-<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+k+1)},p_k,<\bullet>>+<I1_{(r+1)},p_k,<\bullet>>$
$-<I1_r,p_k,<\bullet>>$

Taken into account the place-invariant relationship *PI1*, we obtain $\forall <I1_j,p_k,<\bullet>> \in \Omega^*$
$m(p1)=<I1_{(j+1)}>-<I1_{(j+1)}>+<I1_{(j+k+1)}>+<I1_{(r+1)}>-<I1_r>$

From here and the proposition 1 can be concluded that
$<I1_{(j+k+1)}>=<I1_r>$ and $m(p1)=<I1_{(r+1)}>$ which proves the proposition true.□

Corollary: If there are $k \geq 0$ items stored in the LIFO following the position $<I1_j>$, then
$m(p3)=<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+2)},p_k,<\bullet>>+...+<I1_{(j+k)},p_k,<\bullet>>$
Taken into account the place-invariant relationship *PI1*, then $\forall <Ij,p_k,<\bullet>> \in \Omega^*$
$m(p1).<I1_{(j+k+1)}>=1$
and the first position of the storage, to be unloaded, will be the position $<I1_{(j+k+1)}>$.

Proposition 7: At any time, there is at least one task to be done within the LIFO storage, before it becomes deadlockfree. From the point of view of the model, for any reachable marking of the OCPN model, there is at least one enabled transition in regard to one color from the universal color domain of the net. There is at least one occurrence-mode function with the value true, i.e., 1.

Proof: Each of three feasible states of the storage and all the feasible reachable marking (states) of the OCPN will be considered. And are:

1) Initial state

The LIFO storage is empty, the initial marking of place *p1* is $m_0(p1)=<I1_r>$, that is, the storage loading process begins onto this specific position.

From proposition 1, the place-invariant relationships *PI1, PI2 and PI4*, then
$\forall <I1_i,p_k,<\bullet>> \in \Omega^*$
$m(p1).<I1_r>=<I1_r>.<I1_r>=1;\ m(p3)=0;\ m(p4)=m(p5)=0;\ m(p6)=<\bullet>;$
$m(p2).\Sigma_{1 \le i \le n}<I1_i>=n$
Taken into account these results and the guard of the transitions, it is concluded that only transition *t1* is enabled to fire in regards to the color $<I1_n,p_k,<\bullet>>$.

From the definition 34 only one occurrence-mode function of *t1* is true, i.e., 1, that is
$\gamma_n(t1)=proj(1).m(p2)*abs.m(p6)*[I2_n]=1;\ \forall j \in [1:(n-1)] \Rightarrow \gamma_j(t1)=0.$
Conclusion: The operation "loading a part into the input-port of the LIFO system" is the unique performed action in the system.□

2) *Steady state*

The LIFO system has x ($x \ge 0$) busy positions following the position $<I1_j>$.
From the place-invariant relationships *PI1, PI2, PI4*, the transition-invariant *TI1*, and
$\forall <I1_j,p_k,<\bullet>> \in \Omega^*$

$proj(1).m(p1).<I1_{(j+x+1)}>=1;$
$id.m(p3).[<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+2)},p_k,<\bullet>>+...+<I1_{(j+x)},p_k,<\bullet>>]=x,$
$proj(1).m(p2).[<I1_{(j+x+1)}>+<I1_{(j+x+2)}>+...+<I1_n>]=[n-(j+x)];$
$\{id.[m(p4)+m(p5)]+\ abs.m(p6)\}.<\bullet>=1;$
Taking into account these results, transition *t4* is enabled to fire in regard of all colors
$[<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+2)},p_k,<\bullet>>+...+<I1_{(j+x)},p_k,<\bullet>>] \in \Omega^*$
and transition transition *t1* is enabled to fire in regard of all colors
$[<I1_{(j+x+1)},p_k,<\bullet>>+<I1_{(j+x+2)},p_k,<\bullet>>+...+<I1_n,p_k,<\bullet>>] \in \Omega^*$
From the definition 34 it is possible to conclude that:

- only one occurrence-mode function of *t1* is true, that is
  $\gamma_n(t1)=proj(1).m(p2)*abs.m(p6)*[I2_n]=1;\ \forall i \in [1:(n-1)] \Rightarrow \gamma_i(t1)=0$
  The operation "loading a part into the input-port of the LIFO system" is one of the possible actions to be performed in the system.□
- only one occurrence-mode function of t4 is true, that is
  $\gamma_{j+x}(t4)=proj(1).m(p1)*abs.m(p6)*id.m(p3)=1;\ \forall i \in [j+1:j+(x-1)] \Rightarrow \gamma_i(t4)=0.$□
- both occurrence-mode functions, $\gamma_n(t1)$ and $\gamma_{j+x}(t4)$, correspond to mutual exclusion operations. Then, it is necessary to decide which operation will be performed in the next iteration, i.e., a conflict has to be solved /39/.□

3) *Transitory state*

a) Push operation in advance

The LIFO system has *x* ($x \ge 0$) busy positions following the position $<I1_j>$.

From the place-invariant relationships *PI1, PI2, PI4*, the transition-invariant *TI1* and
$\forall <I1_i,p_k,<\bullet>> \in \Omega^*$
$proj(1).m(p1).<I1_{(j+x+1)}>=1;$
$proj(1).m(p2).[<I1_{(j+x+1)}>+<I1_{(j+x+2)}>+...+<I1_n>]=[n-(j+x)];$
$id.m(p3).[<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+2)},p_k,<\bullet>>+...+<I1_{(j+x)},p_k,<\bullet>>]=x$
$\{id.[m(p4)+m(p5)]+abs.m(p6)\}.<\bullet>=1;\ abs.m(p6)=0;\ id.m(p5)=0.$

From the above considerations and the place-invariant relationship *PI2*, it is concluded that the first element of the marking of p4 must belong to the sub-set of unloaded positions of the system $\Omega 1=\{((\Sigma_{1 \le i \le j}<I1_i>) \cup (\Sigma_{[(j+x+1) \le i \le n]}<I1_i>)),p_k,<\bullet>>\}$

According to this, we have two feasible conclusions $\forall <I1_c,p_k,<\bullet>>> \in \Omega 1$

a1) $\forall\ c > (j+x+1)$ then, $proj(1).m(p2).<I1_{(c-1)}> \ge 1$ and taking into account the definition 34, then

$\gamma_c(t3)=id.m(p4)*proj(1).pred1_1.m(p2)=1$

transition *t3* is enabled to fire in regard of the color $<I1_c,p_k,<\bullet>>$ and a sequence of operations "transfer of parts from one position to the predecessor one" is performed.□

a2) If $c = (j+x+1)$ then, $proj(1).m(p2).<I1_{(c-1)}>=0$ and taking into account the definition 34, then

$\gamma_{(j+x+1)}(t2)=proj(1).m(p1)*id.m(p4)=1$

transition *t2* is enabled to fire in regard of the color $<I1_{(j+x+1)},p_k,<\bullet>>$ and the operation "loading position $<I1_{(j+x+1)}>$" is performed.□

b) Pull operation in advance

The LIFO system has $x\ (x \ge 0)$ busy positions following the position $<I1_j>$.

From the place-invariant relationships *PI1*, *PI2*, *PI4*, the transition-invariant *TI1* and $\forall <I1_i,p_k,<\bullet>> \in \Omega^*$

$proj(1).m(p1).<I1_{(j+x+1)}>=1;$

$proj(1).m(p2).[<I1_{(j+x+1)}>+<I1_{(j+x+2)}>+...+<I1_n>]=[n-(j+x)];$

$id.m(p3).[<I1_{(j+1)},p_k,<\bullet>>+<I1_{(j+2)},p_k,<\bullet>>+...+<I1_{(j+x)},p_k,<\bullet>>]=x$

$\{id.[m(p4)+m(p5)]+abs.m(p6)\}.<\bullet>=1;\ abs.m(p6)=0;\ id.m(p4)=0.$

From the above considerations and place-invariant relationship *PI3* it is concluded that the marking of *p5* must belong to the subset of unloaded storage positions

$\Omega 1=\{((\Sigma_{1 \le i \le j}<I2_i>) \cup (\Sigma_{[(j+x+1) \le i \le n]}<I2_i>)),p_k,<\bullet>>\}.$

From this, we have two feasible conclusions $\forall <I1_u,p_k,<\bullet>> \in \Omega 1.$

b1) $\forall\ u < n$ then, $proj(1).m(p2).<I1_{(u+1)}> \ge 1$ and taking into account the definition 34

$\gamma_u(t6)=id.m(p5)*proj(1).succ(1_1).m(p2)=1$

transition *t6* is enabled to fire in regard of the color $<I1_u,p_k,<\bullet>>$ and a sequence of operations "transfer of parts from one position to the following one" is performed.□

b2) If $u = n$ and taking into account the definition 34

$\gamma_u(t5)=\gamma_n(t5)=id.m(p5).[I1_n]=1,$

transition *t5* is enabled to fire in regard of the color $<I1_n,p_k,<\bullet>>$ and the operation "unloading LIFO system from output-port" is performed.□

Corollary 1: If we suppose that the first "x" positions of the LIFO are loaded, then the initial marking of the net corresponding to this state is:

$m_0=[<I1_{(x+1)}>\ \ (\Sigma_{1 \le i \le [n-(x+1)]}<I1_i>)\ \ (\Sigma_{1 \le i \le x}<I1_i>)\ \ 0\ \ 0\ <\bullet>]^T$

It is concluded that a part can be loaded in the module and transported to position $<I1_{(x+1)}>$ (push-operation) or the part loaded in the position $<I1_x>$ can be unloaded (pull-operation).

Corollary 2: If there is no part stored in the module, and considering that the position $<l1_1>$ of the module has to be loaded and then unloaded in a *push-pull* form, the following sequence of operations must be performed:

- push-operation: loading part $<p_k>$ into the input-port of the LIFO ($<l1_n>$) – transporting part $<p_k>$ to position $<l1_2>$ – loading part $<p_k>$ into goal position ($<l1_1>$)
- pull-operation: unloading part $<p_k>$ from position $<l1_1>$ – transporting part $<p_k>$ to position $<l1_n>$ – unloading part $<p_k>$ from output-port of the LIFO ($<l1_n>$)

Both sequences of operations can be converted into an equivalent sequence of "firing of occurrence-modes of transitions (t,c_t)" in the OCPN model:

- push-operation: $(t1,<l1_n,p_k,<\bullet>>)$, $(t3,R<l1_i>)$, $(t2,<l1_1,p_k,<\bullet>>)$
- pull-operation: $(t4,<l1_1,p_k,<\bullet>>)$, $(t6,R<l1_i>)$, $(t5,<l1_n,p_k,<\bullet>>)$

According to definition 34, the evolution of the OCPN model, which corresponds to the last sequences, can be represented by means of the following set of occurrence-mode functions:

- *push-pull* operation: $\gamma_n(t1)$, $\gamma_j(t3)_{\forall j\in[2:n]}$, $\gamma_1(t2)$, $\gamma_1(t4)$, $\gamma_j(t6)_{\forall j\in[1:(n-1)]}$, $\gamma_n(t5)$

Axiom 6:For the sake of simplicity, we restrict our work to simplified resource models. This means that we do not consider certain more complex structures as, for example, the case of two manipulators as parts of one resource. The Ordered Color Petri net-based models of a resource generated in this work are then "persistent or deterministic" Petri net models (e.g., a Petri net structure without conflicts) /74/.

Remark: According to last consideration, the OCPN models of resources are characterized by the existence of one, and only one, transition-flow. Each OCPN model of resource is then a mono transition-flow net /9/.

Proposition 8: Transitions *t1* and *t5* have attached only the occurrence-color $<l1_n,p_k,<\bullet>>$. They model the beginning of the push-operation and the end of the pull-operation.

Proof: Immediate from structure of transition-invariant *TI1* and the guard attached to each of them.□

Proposition 9: Transitions *t3* and *t6* have attached a single occurrence-color from the universal color domain of the net. At a time, only one color is enabled to fire. They model the "transfer" operation between every two consecutive storage positions during the push- and pull-operations, which are performed in mutual exclusion.

Proof: Immediate from structure of transition-invariant *TI2* and *TI3* and proposition 5.□

Proposition 10: Transitions *t2* and *t4* have attached a set of occurrence-colors, which coincides with the universal color domain of the net. *t2* models the ending of push-operation / loading of a specific dynamic queue position, and *t4* models the beginning of a pull-operation / unloading of a specific dynamic storage position.

Proof: Immediate from structure of transition-invariant *TI1*.□

Proposition 11: The movement of a part $<p_k>$ in the LIFO module (loading and unloading of the part) imposes a sequence of *push-pull* operations. This is possible only if the OCPN model has repetitive or cyclic behavior.

Proof: By considering the results presented in proposition 1 and its corollaries, $\forall <I1_i, p_k, <\bullet>> \in \Omega^*$ and an initial marking

$M_0 = [<I1_{(x+1)}> \ (\Sigma_{1 \le i \le [n-(x+1)]} <I1_i>) \ (\Sigma_{1 \le i \le x} <I1_i>) \ 0 \ 0 <\bullet>]^T$

the sequence of push-pull operations corresponds to an evolution of the OCPN, which can be expressed by means of the following set of occurrence-mode functions

$\gamma_n(t1), \gamma_j(t3)_{\forall j \in [(x+2):n]}, \gamma_{(x+1)}(t2), \gamma_{(x+1)}(t4), \gamma_j(t6)_{\forall j \in [(x+1):(n-1)]}, \gamma_n(t5)$

The last sequence can be converted into an equivalent occurrence/firing sequence of the OCPN model which firing count vector Θ matches with the transition-flow *tf1*.

According to definition 17, equation 2.11 (see chapter 2)

$M'(p) = M_0(p) + I.\Theta$

Since $I.\Theta$ coincides with $TI1$, it is concluded that $M'(p) = M_0(p)$. The part was loaded and then unloaded from the LIFO module, which proves the proposition true.□

Conclusions

All the places of the OCPN model take part at least in one place-invariant relationship and each place has a bound; the OCPN model is bounded.

By analyzing all the obtained place-invariant relationships, it is possible to ensure that the OCPN model cannot have deadlocks.

It has been demonstrated that the dynamic LIFO storage is reversible for the given initial marking.

The application of the above addressed methodology allows modeling and validation of the specifications of each resource of a flexible production system. However, neither the structure of the whole system, nor the constraints imposed from this structure to the behavior of each resource, were considered.

3.3.3 Validation of System Layout Specifications

Even if the different modules, e.g., resources, of the system are correctly modeled and validated during the first phases of the design, it is necessary to perform the analysis and the validation of specifications of the complete model of the system, since the interactions between the different modules can give rise to the main properties of the whole system /32/. The last addressed properties are those that exist as a result of the interaction of different modules, and that cannot easily be determined from the independent validation of the component modules.

Since the construction of the OCPN model of the whole system is preceded by the specification of a set of scenarios, which the intended system should obey to, the results of the actual analysis phase must help the designer to validate all these specifications.

The validation of the specifications of the system's layout can be viewed as a "recognition" or "verification-validation" task, where the input is the particular arrangement of

OCPN models of resources, and the task is to verify that the OCPN model of the whole system actually matches the production and material-flow purposes.

The main result of the modeling approach described in section 3.2.3 is the possibility of composing analysis results of the individual modeled resources, in order to obtain results which are valid for the entire OCPN model. Using the information from this analysis, it is straightforward to validate the specifications of the whole system, such as:

- boundedness, conservativeness, deadlockfreeness, reversibility, etc.;
- reachability of some special state;
- mutual exclusion among states of resources;
- capacity of storage zones;
- sequence of modeled operations;

For this purpose, a constructive proof of how place- and transition-flows of the modeled resources can be combined to place- and transition-flows of the OCPN model of the whole system is presented as follows.

Main goal is to construct place- and transition-invariant relationships of the OCPN of the system from the place- and transition-invariant relationships of the models of the individual resources and then to validate the behavioral specifications of the whole system.

The flexible production cell of Fig. 17 serves as an example. It has 3 resources, and three kinds of information are basically considered for describing their static and dynamic specifications, they are the basic color domains of the OCPN model of Fig. 18.

Fig. 19 shows the individual OCPN models of the resources of the exemplary FPC and the corresponding place-flows and place-invariant relationships. For a better understanding of the meaning of each invariant relationship, Fig. 20 depicts the same results in a graphic form.

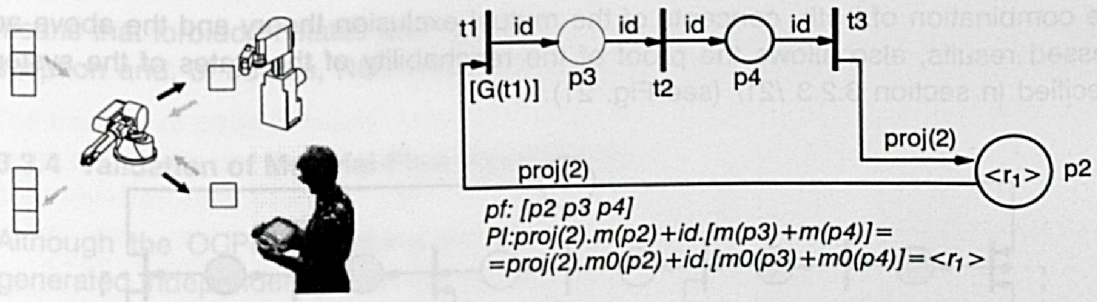
For instance, Fig. 19(a) and 20(a) show the meaning of the place-flows in relation with the specifications of the robot 1 ($<r_1>$) when considering it embedded in the layout of the whole flexible production cell. Similar results can be issued by analyzing the models of the other resources.

An overview of these graphical descriptions allows proofing that the set of place-flows (i.e., place-invariant relationships) of the models of resources can be combined to cover the total system.

We are prepared now to apply the same formal methodology used in section 3.3.2.

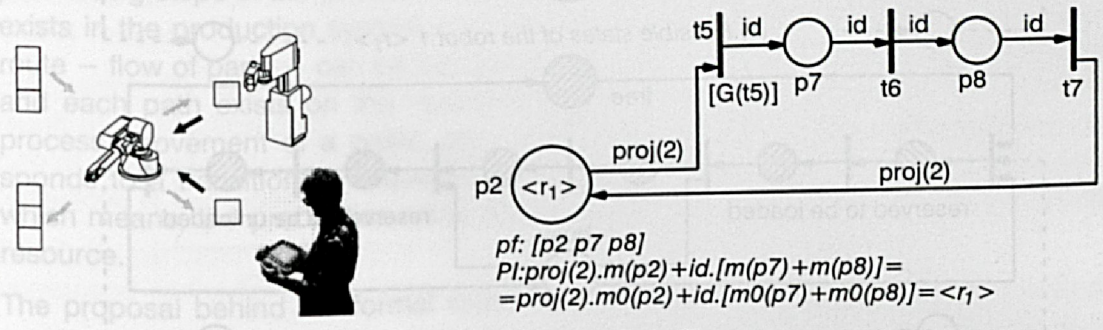
Conclusions

All places of the OCPN models take part at least in one place-invariant relationship. It means that each place has a bound, which means the OCPN models are bounded and the model of the whole system too.



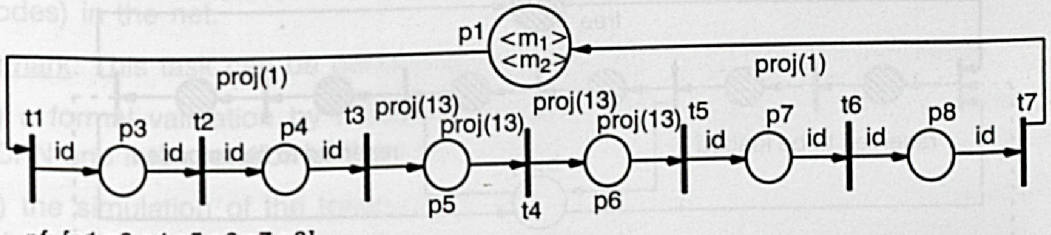
pf: [p2 p3 p4]
PI:proj(2).m(p2)+id.[m(p3)+m(p4)]=
=proj(2).m0(p2)+id.[m0(p3)+m0(p4)]=<r1>

(a) Robot loads work positions



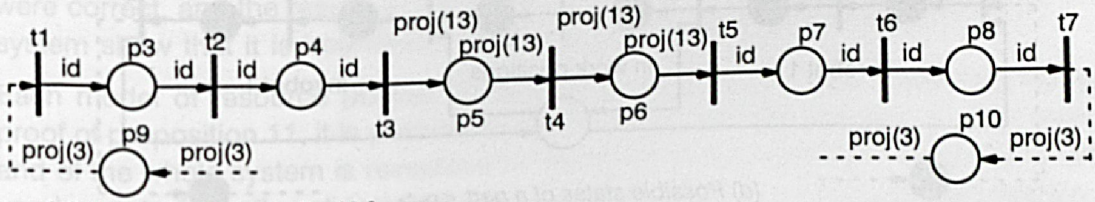
pf: [p2 p7 p8]
PI:proj(2).m(p2)+id.[m(p7)+m(p8)]=
=proj(2).m0(p2)+id.[m0(p7)+m0(p8)]=<r1>

(b) Robot unloads work positions



pf: [p1 p3 p4 p5 p6 p7 p8]
PI1:proj(1).m(p1)+id.[m(p3)+m(p4)+m(p7)+m(p8)]+proj(13).[m(p5)+m(p6)]=<m1>
PI2:proj(1).m(p1)+id.[m(p3)+m(p4)+m(p7)+m(p8)]+proj(13).[m(p5)+m(p6)]=<m2>

(c) Robot 2 and human operator



pf: [p3 p4 p5 p6 p7 p8 p9 p10]
PI1:proj(3).[m(p9)+m(p10)]+id.[m(p3)+m(p4)+m(p7)+m(p8)]+proj(13).[m(p5)+m(p6)]=0

(d) Parts to be processed

Fig. 19: Structural Analysis of the OCPN Models of Resources of the FPC in Fig. 18

By analyzing all the place-invariant relationships obtained, it is possible to ensure that the OCPN models cannot have deadlocks when they work together.

The combination of both, concepts of the mutual exclusion theory and the above addressed results, also allows the proof of the reachability of the states of the system specified in section 3.2.3 /21/ (see Fig. 21).

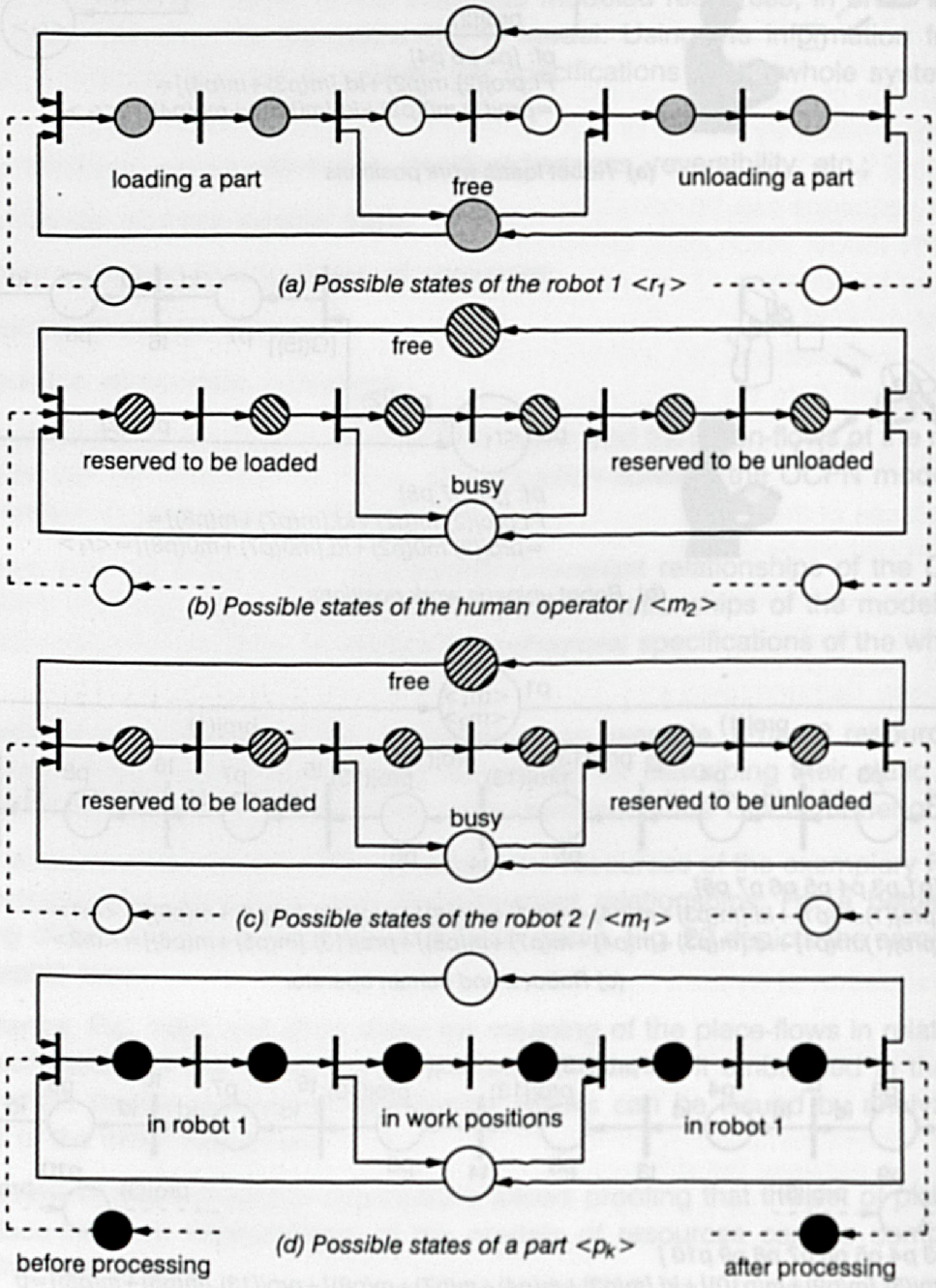


Fig. 20: Validation of Specifications for the FPC's Layout proposed in Fig. 17

The main goal is then to show "what is where to be done" in the production environment using the structural information of the OCPN model.

If the H-L-PN was developed correctly, all possible states as specified above are described formally by means of the place-flows (i.e., place-invariant relationships). This

means that forbidden states and operations are not accepted by the mathematical description and, of course, NOT ACCEPTED during the operation of the real system.

3.3.4 Validation of Material-Flow Specifications

Although the OCPN describing the structure - layout - of the production system is generated independently from the material-flow specifications (e.g., production routes), the modeling methodology addressed in the last section shows that there is a connection between the model and these specifications in reality. This is because each of the processing steps of the production routes has to be carried out on a component which exists in the production system and was modeled previously. Therefore, a production route – flow of parts – can be thought of as a path through the manufacturing system, and each path exists on the modeling level. Every processing step (i.e., machining process, movement of a pallet, etc.) from the production route specifications corresponds to a transition or sequence of transitions, to be fired from the OCPN model, which means, the production route action is executed by the corresponding modeled resource.

The proposal behind the formal specification of material-flow in a flexible production system, using the information provided by the OCPN models, is that each flow of parts can be mapped to one, and only one, sequence of firing of transitions (i.e., occurrence-modes) in the net.

Remark: This task can be performed by using two possible methodologies:

(a) a formal validation by means of the structural analysis of transition-flows of the OCPN and interpretation of them with regard to the material-flow specifications, and/or

(b) the simulation of the token-game of the Ordered Colored Petri net-based model enhanced with timing parameters (i.e., reachability-coverability graph analysis).

Formal Validation

Let us assume that the results of the validation of the OCPN models of the resources were correct, and the results of the structural analysis of the OCPN model of the whole system show that it is bounded, conservative and live.

Each model of resource posses at least one transition-flow and, applying the same proof of proposition 11, it is possible to demonstrate that the behavior of each resource and of the whole system is reversible for the given initial marking. This property will be used now to formally validate material-flow specifications of the modeled system.

In order to attain a better formal description of the material-flow specifications, there arises the concept of transition-supports (t-supports/ts) of the OCPN model (see chapter 2, definition 31).

The point here is that the evolution of the OCPN for a given sequence of firing of transitions can be formally expressed taking into consideration the structure of the corresponding transition-support. The use of the information provided by each transition-support combined with the structural information of the firing-modes of the transitions,

i.e., guards associated to some of them, provides an opportunity to exploit both to produce a set of transition-flows.

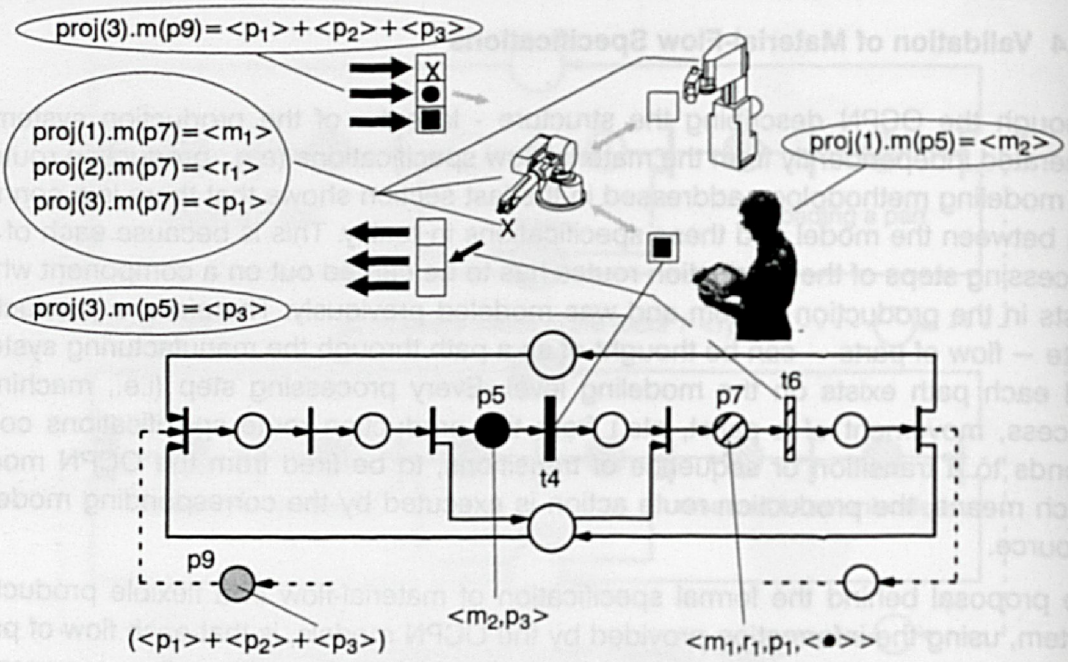


Fig. 21: Validation of States of Resources using Structural Analysis of the Model

The material-flow specifications of the flexible production cell described in section 3.3.2 serves as an example in order to make the idea behind the validation of material-flow specifications using the information provided by the transition-supports more clearly. Fig. 22 shows same production-paths of the cell and their mathematical description by means of transition-supports and transition-flows of the OCPN model.

Proposition 12: The movement of a part $<p_k>$ in the modeled production environment for a given production-path corresponds to an equivalent occurrence/firing sequence of the OCPN model, which firing counter vector Θ matches with one of the transition-flows of the net. Then, for each production-path/material-flow, which is to be formally specified in the production environment, a transition-flow has to be found in the OCPN coordination model. With this transition-flow can be validated a repetitive or cyclic behavior of the net for the proposed production-path.

Proof: Let us consider the production-path depicted in Fig. 22(c) as the material-flow specification which is to be validated.

At this point, taking into account the definition 30, section 2.5.4, chapter 2, the transition-flow can be converted into an equivalent firing count vector Θ, then
$\forall\ \omega^* \in \Omega^*, \omega^* = \{<m_i, r_j, p_k, <\bullet>>\}$ an the initial marking proposed in Fig. 18
$M'(p) = M_0(p) + I.\Theta$
Since $I.\Theta$ is a transition-invariant, it is concluded that $M'(p) = M_0(p)$.

As main conclusion, the evolution of the net according to a firing sequence matching with the transition-flow, validates the movement of a part according to the specified

production-path in the manufacturing environment and a cyclic / repetitive behavior of the model, which proves the proposition true.□

The transfer of parts between two consecutive resources in the manufacturing environment occurs as one indivisible operation in which the preservation of specifications of both involved resources is a main goal. According to the modeling method presented in this work, this operation is modeled by means of only one transition.

Taking into account the results above presented, this transition takes a part of the transition-support structures, corresponding to the OCPN models of the involved resources. Therefore, the enabling-condition and the firing of this transition occurs as one indivisible action sharing the values assigned by the occurrence-colors of the involved OCPN models.

Proposition 13: The material-flow in a manufacturing environment can be seen as the result of production-path's composition of involved individual components. The structure of a transition-support and related transition-flow, which formally specifies this material-flow, is composed of the transition-supports of the individual OCPN models.

Proof: By considering the same example, the production-path specified in Fig. 22(c) is composed of the following basic paths:

Basic $path_1$: Robot 1 transfers one part 3 from the input position to the human operator.
Basic $path_2$: Human operator processes the part.

Basic $path_3$: Robot 1 transfers the processed part from the human operator to the output position.

Each of the above addressed basic paths can be formally expressed by the following basic transition-supports of the OCPN models of the involved components:
Basic $path_1$ -> Basic T-$Support_1$: t1,t2,t3 -> evolution of the OCPN model of Robot 1 (loading a part)
Basic $path_2$ -> Basic T-$Support_2$: t5,t6,t7 -> evolution of the OCPN model of Robot 1 (unloading a part)
basic $path_3$ -> basic T-$Support_3$: t1,t2,t3,t4,t5,t6,t7 -> evolution of the OCPN model of Human operator

Analysis of the composition of the transition-support in Fig. 22(c) allows to find the above addressed basic transition-supports, which proves the proposition true.□

Proposition 14: If a transition models "transfer of parts and information between resources" (an input- /output-port), the sum of colored tokens consumed by the firing of the transition is equal to the sum of colored tokens produced by this firing (parts- and information-flow preservation). The sum of the colored markings before and after the firing of the transition is constant.

Proof: According to the definitions of place-invariant relationships presented in section 2.5.4 and taking into account the results of proposition 13 it is possible to ensure that:

i) Flow of parts between two connected resources is modeled by the firing of transition which belongs to two basic transition-supports, each one modeling the behavior of each involved resource. Firing of this transition produces the movement of a colored

marking between places modeling states of resources. These places take part of a place-invariant relationship of the OCPN model.

ii) Flow of information between two connected resources is modeled by firing of the same transition, but now its firing produces the transfer of another colored marking between places which model states of the involved resources. These places take now part of two others place-invariant relationships of the same OCPN model.□

As an example, let us consider the transition t3 which models transfer of parts from Robot 1 to Robot 2 or human operator (see Fig. 22(d)). This operation involves the output-port of Robot 1 and input-port of Robot 2 or human operator. The structural analysis of the OCPN models of both resources shows that this transition takes part of two transition-supports (basic T-Support$_1$ and basic T-Support$_3$) addressed in proposition 13. Firing of t3 consumes the marking color ω^* from p4 (flow of part and information), produces the marking $<r_1>$ of p2 (flow of information) and produces the marking $<m_2,p_3>$ from p5 (flow of parts).

Firing of t3 belonging to the T-Support addressed in Fig. 22(c) produces the movement of the marking ω^* (e.g., id.m(p4)) from place p4, of the marking $<r_1>$ to place p2 (e.g., proj(2).m(p4)), and of the marking $<m_2,p_3>$ to p5 (e.g., proj(1,3).m(p4)).

- The markings of places p4 and p5 are in mutual exclusion regarding the marking $<m_2,p_3>$. Flow of parts: a part can be found in resource "robot 1" or in resource "human operator".

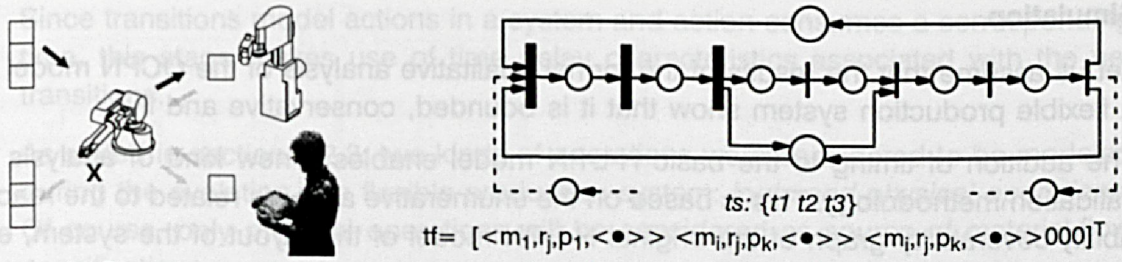
- The markings of places p4 and p2 are in mutual exclusion regarding the marking $<r_1>$. Flow of information: robot 1 can be found busy by a part of type 3 or free.

It is important to observe that the change of the type of tokens (colors) on the places caused by the occurrence of the transition t3 does not depend only on the current marking. Instead, it is completely determined by the structure of the net. Therefore, in order to keep track of the distribution of tokens while the transition fires, it suffices to consider the relative changes for every place connected to the transition.□

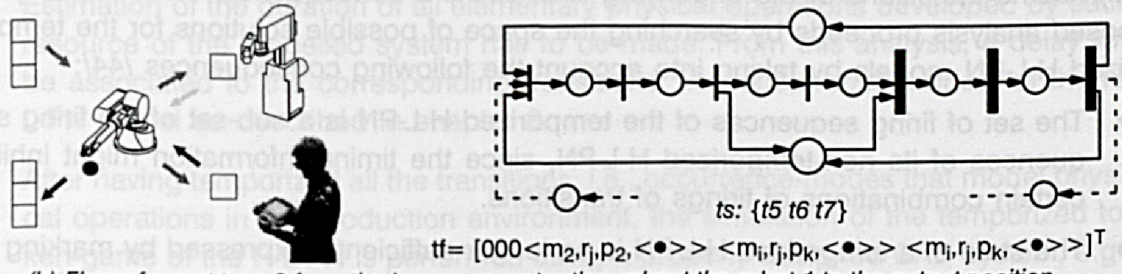
Conclusions

The formal validation methodology proposed in this section can be used to compute all the production-paths for a given arrangement of resources of a flexible production system. With the presented formal specification, the designer of flexible production systems is able to calculate, validate and specify, in advance

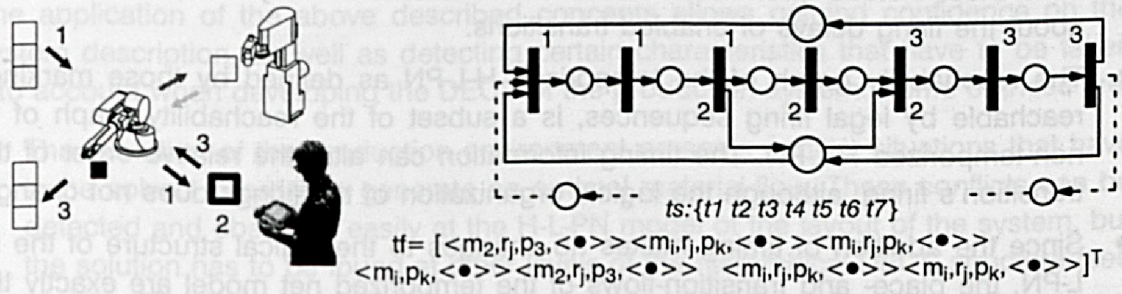
- the total number of travel routes of pallets and exchange of parts and information between resources of the system on the basis of the structural properties of the OCPN models (e.g., each production-path can be represented by only one transition-support and related place-invariant relationships;
- the behavior of each resource is represented by one basic transition-support and related place-invariant relationship).



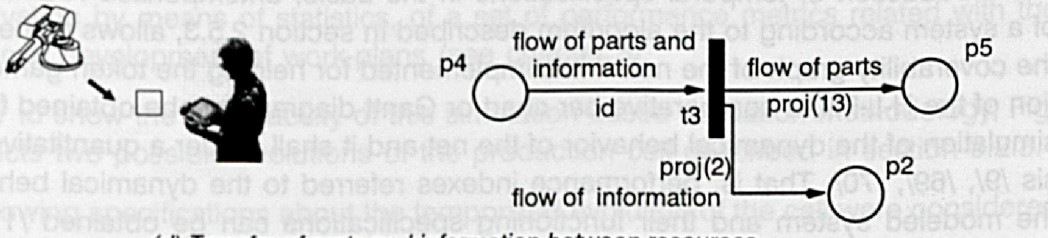
(a) Flow of a part type 1 from the input position throughout the robot 1 to the robot 2



(b) Flow of a part type 2 from the human operator throughout the robot 1 to the output position



(c) Flow of a part type 3 from the input position throughout the robot 1 and the human operator to the output position



(d) Transfer of parts and information between resources

Fig. 22: Validation of Material-Flow Specification using Structural Analysis of H-L-PN

Nevertheless, the main difficulty, which is to be solved for a better understanding of the evolution of the modeled system, is that transition-supports, derived transition-flows, and place-invariant relationships of the OCPN model do not provide full information about the causal relationship between transition occurrences, (i.e., causal relationship between actions and states in the system). Therefore, after the formal validation has been done it is valuable to perform the simulation of the token-game of the net. In this case, each step of a production-path can be validated in the OCPN model taking into account the causality relationship among them.

Simulation

Let us assume that the results of the formal qualitative analysis of the OCPN model of a flexible production system show that it is bounded, conservative and live.

The addition of timing to the basic H-L-PN model enables a new kind of analysis – validation methodology. This is based on the enumerative analysis related to the reachability-coverability graph of the original H-L-PN model of the layout of the system, enhanced with time related specifications of the flexible production system associated with some transitions of the underlying H-L-PN. As a consequence, the herewith proposed analysis proceeds by searching the space of possible solutions for the temporized H-L-PN models by taking into account the following consequences /44/:

- The set of firing sequences of the temporized H-L-PN is a sub-set of the firing sequences of its non-temporized H-L-PN, since the timing information might inhibit certain combinations of firings of transitions.
- The state of a temporized H-L-PN is no longer sufficiently expressed by marking of its non-temporized H-L-PN. Depending upon the particular temporal model, the state must include information about the transitions already fired, or information about the firing delays of enabled transitions.
- The reachability graph of the temporized H-L-PN as defined by those markings reachable by legal firing sequences, is a subset of the reachability graph of its non-temporized H-L-PN. The timing information can alter the relative order of the transition's firing, although the logical organization of the firings does not change.
- Since the addition of timing makes no changes to the logical structure of the H-L-PN, the place- and transition-flows of the temporized net model are exactly the same as for its non-temporized H-L-PN.

The introduction of temporal specifications in the basic, untemporized H-L-PN model of a system according to the algorithm described in section 2.5.3, allows refinement of the coverability graph of the net. If it is implemented for helping the token-game evolution of the H-L-PN, a comparative bar chart or Gantt diagram can be obtained from the simulation of the dynamical behavior of the net and it shall render a quantitative analysis /9/, /69/, /70/. That is, performance indexes referred to the dynamical behavior of the modeled system and their functioning specifications can be obtained /17/, /22/. Detailed discussion about this kind of analysis is beyond the scope of this work. The referenced bibliography can be consulted for more information.

For the purposes of this section, i.e., the validation of material-flow specifications, the Gantt diagram obtained from the temporized evolution of the H-L-PN model of a flexible production system is a very good source of information. However, this validation method can be applied only if the following points are also considered during the modeling phase:

- Some sort of additional information must be available beforehand on the elementary operations which are being modelled, for instance, the main operative characteristics of the resources of the system. From this previous knowledge there arise the initial values for temporizing the original developed underlying H-L-PN.

- Since transitions model actions in a system and action consumes a corresponding time, this stage makes use of time delay characteristics associated with the net transitions.
- As stated in section 3.2.3, two kinds of operations were considered to be modeled during the evolution of a flexible production system: *logic and physical operations.* Of course, only physical operations will be considered as source of material-flow specifications.
- Estimation of the duration of all elementary physical operations developed by each resource of the modelled system has to be made. From this analysis, a delay will be associated to the corresponding transitions, i.e., occurrence-modes, of the H-L-PN model as described in chapter 2.
- After having temporized all the transitions, i.e., occurrence-modes that model physical operations in the production environment, the simulation of the temporized token-game of the H-L-PN is performed using a properly designed CAD system, e.g., simulator.

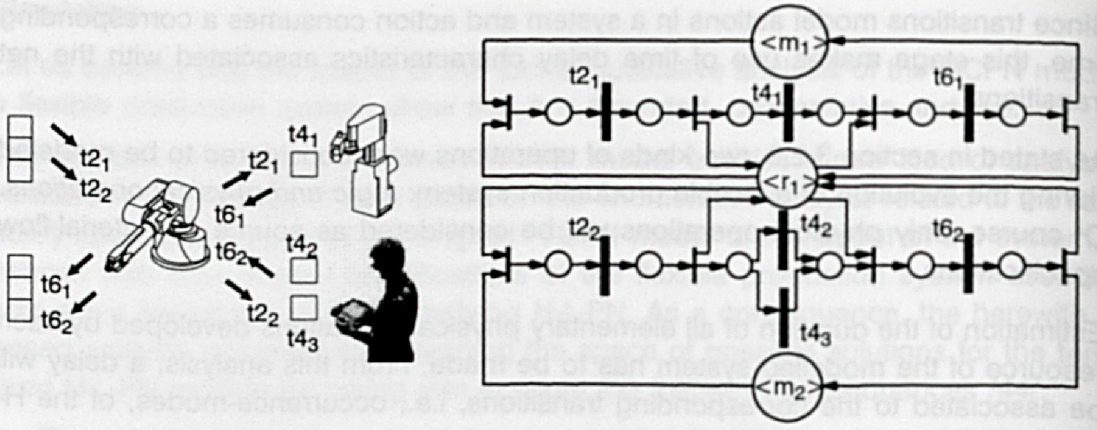
The application of the above described concepts allows gaining confidence on the system description, as well as detecting certain characteristics that have to be taken into account when developing the DECS of the production system. Some of these are:

- The evolution of the production environment presents conflict situations that have to be solved in order to generate an optimal material-flow. These conflicts can be detected and observed easily at the H-L-PN model of the layout of the system, but the solution has to be found at other levels of the hierarchical DECS structure (see chapter 5).
- The material-flow specifications depicted in the Gantt diagram can be used for the derivation by means of statistics, of a set of performance metrics related with the optimal development of work-plans (see chapter 5).

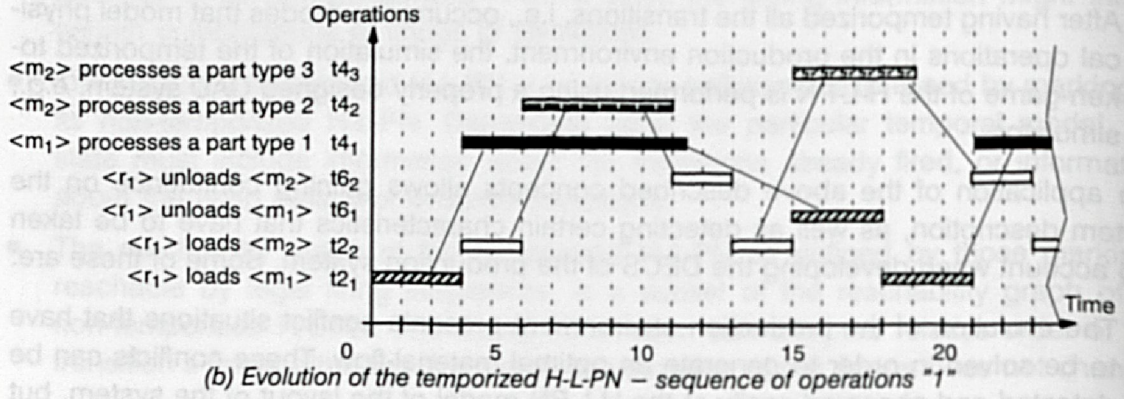
In order to show the applicability of this simulation-based validation methodology, Fig. 23 depicts two possible evolutions of the production cell described in section 3.2.3.

The following specifications about the temporized evolution of the cell were considered and modeled by means of temporized transitions:

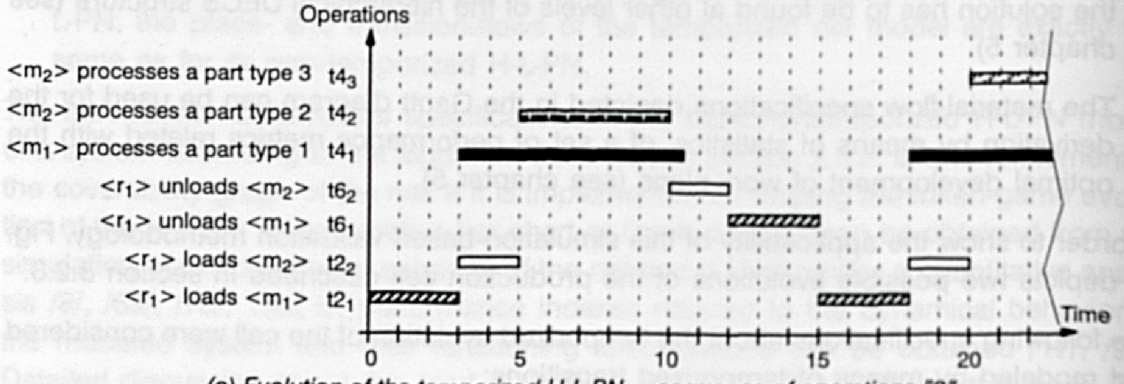
- Robot 1 loads work position of the robot 2: *3 [time units]*
- Robot 1 loads work position of the human operator: *2 [time units]*
- Robot 1 unloads work position of the robot 2: *3 [time units]*
- Robot 1 unloads work position of the human operator: *2 [time units]*
- Robot 2 processes a part type 1: *7,5 [time units]*
- Human operator processes a part type 2: *5 [time units]*
- Human operator processes a part type 3: *4 [time units]*



(a) H-L-PN coordination model enhanced with timing specifications



(b) Evolution of the temporized H-L-PN – sequence of operations "1"



(c) Evolution of the temporized H-L-PN – sequence of operations "2"

Fig. 23: Validation of Material-Flow Specifications using Simulation of H-L-PN Models

For better understanding of the example, the H-L-PN of Fig. 18 is represented by means of equivalent *Temporized Petri Net (TPN)* as defined in chapter 2. Each occurrence-mode of transition of the H-L-PN that models a physical operation in the production environment is represented by means of temporized transition of the TPN. The occurrence-modes of transitions that model logic operations are represented by means of immediate transitions of the *TPN*. Motived by these modeling's restrictions, and without considering any difference between the three possible operations *"robot 1 loads*

work positions of robot 1 or human operator", Fig. 23(a) presents, for example, the transition *t2* of the H-L-PN of Fig. 18 as a pair of temporized transitions $t2_1$ and $t2_2$.

Since the simulation of the temporized evolution of the TPN of Fig. 23 imitates the behavior of the flexible production cell as it evolves over time, basic for the approach of this simulation is the existence of specifications about work-plans to be developed in the system, priorities for performing particular operations, etc. These specifications help solving conflicts and defining wished sequence of operations. Fig. 23(b) and (c) depicts the above concepts by means of two possible temporized evolutions of the exemplary cell.

Conclusions

The material-flow specifications of a flexible production system can be perfectly validated by means of the simulation of the token-game of a temporized H-L-PN model. For this purposes, the H-L-PN model of the layout of the system is enhanced with the attributes and properties of the Temporized Petri Nets (TPN) defined in chapter 2. Nevertheless, for a given layout, it is very important to consider that the temporization of the transitions of the model, i.e., occurrence-modes, must be compatible with the timing characteristics of the systems's resources, and the temporized evolution of the net with the specifications of work-plans, which are to be developed in the system.

3.4 Combining Modeling and Validation Approaches for Design of Flexible Production Systems

Combining modeling and validation methods mentioned in sections 3.2 and 3.3, the research of this section is dedicated to development of theoretical foundations and to presentation of a practical procedure of formally FPS designing with H-L-PN. The idea is to get useful information about structure and behavior of the FPS by reasoning on the structure and behavior of its H-L-PN-based model (i.e., validation of specifications). In other words, the model reflects the set of specifications of the FPS by means of its properties. Moreover, the synthesis of such a formal model can be done by taking into account that it fulfils a set of properties because of the modeling requirements considered during its development.

The integration of modeling and validation is based on the following main result /21/, /117/: *"Only having, among others, the properties boundedness, conservativeness, deadlockfreeness, reversibility, the H-L-PN model will be seen as a safe formal specification of a flexible production system".*

An analysis of results of the simulation approach presented above shows that an exhaustive (complete) validation of the FPS specifications requires that the properties of the model are proved under all possible conditions. As depicted in Fig. 23, combinations of feasible values of model variables, e.g., enabling condition of transitions, can generate many logical paths in the model execution (different behaviors of the FPS). Due to time and budgetary constraints, it is impossible to validate the accuracy of all logical paths. Therefore, the purpose is to increase the confidence in model credibility

as much as by its formal properties, i.e., properties of its structure, rather than trying to test the model after its conception.

The approach proposed here relies on the fact that the FPS structure and behavior specifications have to be formally represented by means of place- and transition-flows of the H-L-PN model. Fig. 24 depicts the results of applying this formal synthesis method to a sample flexible assembly cell (more details about this flexible production cell can be found in /38/).

To help ground the methodology, Fig. 24(a) depicts the desired system and production specifications. The H-L-PN-based model of the cell is developed taking into consideration the characteristics of its components, e.g., transport system structure and strategies, robots, etc., and the set of possible material-flow specifications. Fig. 24(b) and 24(c) show the H-L-PN-based formal description, of the desired production-path specification together with the involved resources, by using structural properties of the net. So, each set of operations is formalized by means of a transition-support and the possible states of the involved resources by means of place-supports.

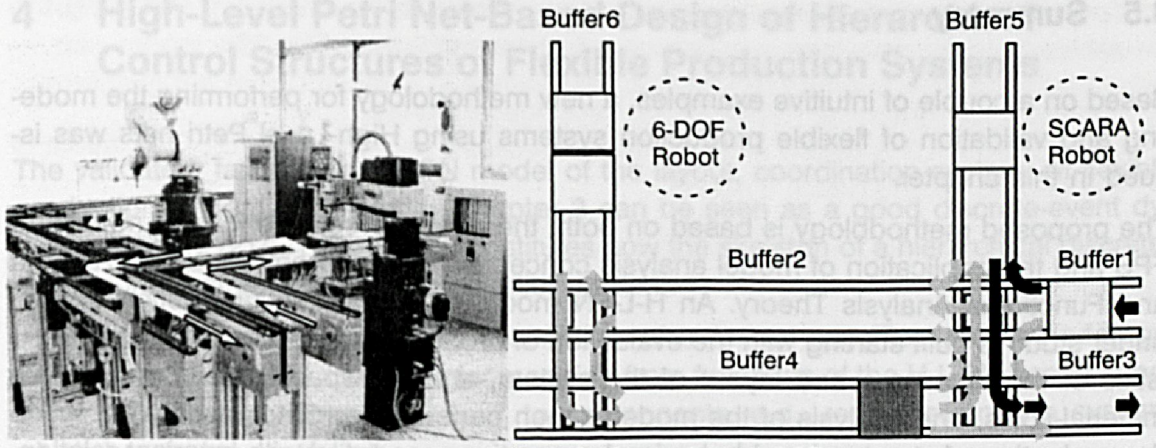
- Movement of a pallet for the specified production-path (T-Support : *t1 t2 t4 t6 t7 t9 t10 t11*)
- Movement of pallets on buffer1 (T-Support : *t1 t2 t4*)
- Movement of pallets on buffer5 and the work of robot on it (T-Support: *t4 t6 t7*)
- Movement of pallets on buffer3 (T-Support: *t7 t9 t10*)
- Location of a pallet (P-Support: *p2 p3 p5 p6 p8*)
- States of transport-places in buffer1 (P-Support: *p1 p2 p3*)
- States of transport-places in buffer5 (P-Support: *p4 p5 p6*)
- States of transport-places in buffer3 (P-Support: *p7 p8*)

The flow of parts and information at the production level will be formalized by means of composition of transition-supports as depicted in Fig. 24(d) and 24(e) respectively.

It should be noted here that the resulting H-L-PN is by no means a complete formal description of the real production environment. It is rather a skeleton that constitutes a first iteration through the FPS development. Nevertheless, it offers the following important advantages:

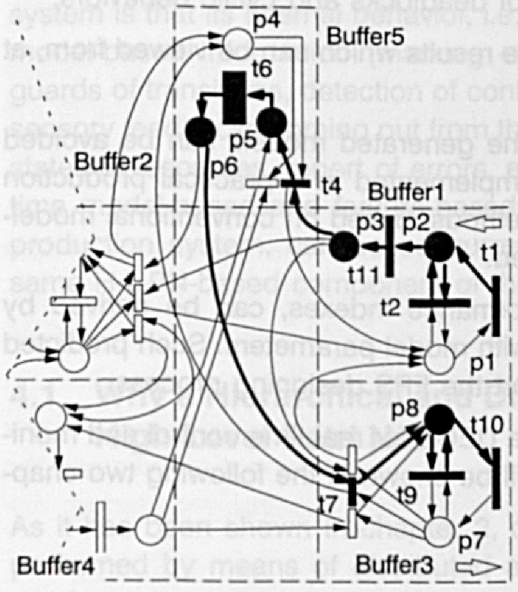
- The resulting structures and behaviors do not contradict the specifications: the model is correct by construction.
- The formalized specifications are maximally permissive: all states and operations which do not contradict the specifications, and are contained within a place- or transition-support, are in the reality allowed to happen.

As main result, the production engineer has a H-L-PN-based virtual production environment which is a faithful representation of the real system and behaves according to the desired production specifications.

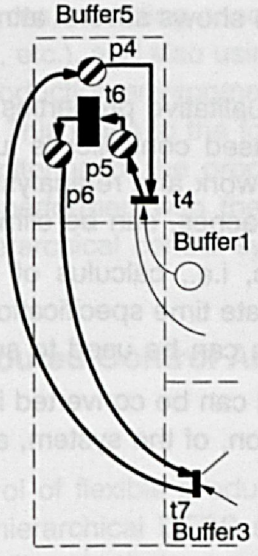


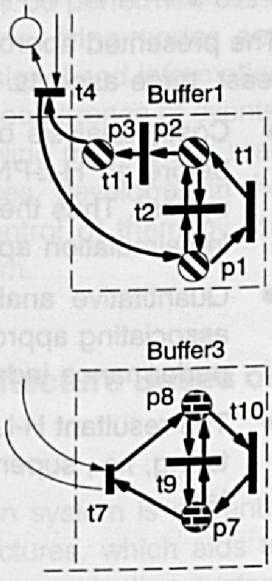
Specification ➡ production-path/material-flow:
Input/Movement in buffer1 – > Movement in buffer5 – > Movement in buffer3/Output

(a)

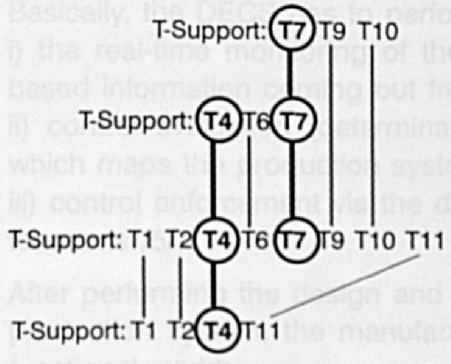


(b)



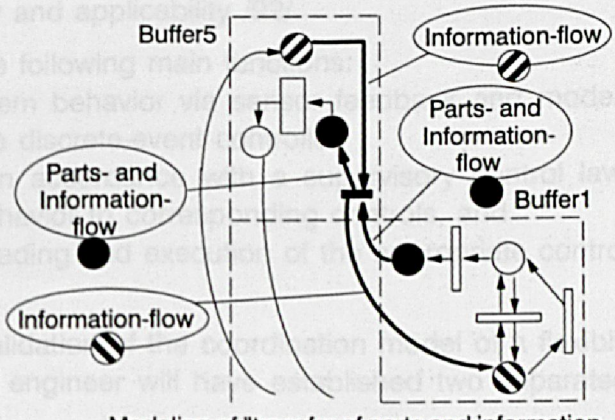


(c)



Composition of production-path specifications

(d)



Modeling of "transfer of parts and information between resources"

(e)

Fig. 24: Integration of H-L-PN-based Modeling and Validation to Design FPS

3.5 Summary

Based on a couple of intuitive examples, a new methodology for performing the modeling and validation of flexible production systems using High-Level Petri nets was issued in this chapter.

The proposed methodology is based on both, the design of reliable H-L-PN models of FPS and the application of model analysis concepts extracted from the Linear Algebra and Functional Analysis Theory. An H-L-PN model of the flexible production system under study is built starting with the evaluation of products to be processed, characteristics of the resources and layout of the system, and the specifications of possible material-flows. The analysis of the model is then performed which is mainly based on the evaluation of structural and behavioral properties associated with invariant relationships. The results of the analysis are finally used to validate the model in relation with, among others, resource capacities, avoidance of deadlocks and cyclic behaviors.

The presented approach shows several attractive results which can be viewed from, at least, three aspects.

- Costly analysis of qualitative properties of the generated models can be avoided before an H-L-PN-based controller is fully implemented in a practical production system. Thus the re-work and re-analysis methods, based on conventional modeling-simulation approaches, can be eliminated.
- Quantitative analysis, i.e., calculus of performance indexes, can be derived by associating appropriate time specifications with model parameters. Such predicted performance indexes can be used to support the FPS designing process.
- The resultant H-L-PN can be converted into a DECS for real-time control and monitoring, i.e., supervision, of the system, as will be shown in the following two chapters.

4 High-Level Petri Net-Based Design of Hierarchical Control Structures of Flexible Production Systems

The validated, failure-free H-L-PN model of the layout, coordination system, of flexible production systems achieved in chapter 3 can be seen as a good discrete-event dynamic scheme /21/, /117/, and it constitutes now the skeleton of a hierarchical discrete-event control structure of the production system.

In this chapter is presented the description of a formal specification methodology, which represents the knowledge for mapping from functions of the H-L-PN-based specification of the coordination system, to components, e.g., resources, of the system, which has to be controlled.

Important consideration about the H-L-PN-based hierarchical discrete-event control system is that its internal behavior, i.e., token-game of the net, will be performed using model-based information (marking of places, enabling-conditions, firing-modes and guards of transitions, detection of conflicts, etc.), and also using signal and information sensory feedbacks coming out from the production environment (occurrence of events, states of resources, report of errors, etc.). This leads to the following main result, real-time model-based and feature-based monitoring of the processes, developed in the production system, can be concurrently performed with the control of them by the same H-L-PN-based component of the hierarchical control system.

4.1 Why a Hierarchical and Distributed Control Architecture Based on High-Level Petri Net?

As it has been shown in chapter 2, control of flexible production system is currently performed by means of distributed and hierarchical DECS structures, which aids to resolve three main critical issues in controlling the behavior of the production system on the *shop floor*: flexibility, complexity and applicability /92/.

Basically, the DECS has to perform the following main functions:
i) the real-time monitoring of the system behavior via sensor feedback and model-based information coming out from the discrete-event controller,
ii) control evaluation (determination) in accordance with a supervisory control law, which maps the production system behavior to corresponding controls, and
iii) control enforcement via the downloading and execution of the appropriate control functions /5/.

After performing the design and the validation of the coordination model of a flexible production system, the manufacturing engineer will have established two separated functional worlds:

- Real production world (production environment)
  The specifications of a set of hardware components, i.e., machines, robots, conveyors, etc, capable of providing the wished production specifications, and the corre-

sponding layout of the flexible production system to be controlled including the sensor/actuator interface, e.g., process interface (PI) (see Fig. 25);

- Mathematical-graphical model of the production world
  A validated H-L-PN-based model of each hardware component and of the layout of the system (coordination system).

However, neither a structure for the generation of the control and monitoring functions defined above, nor a correct formalism for describing a close interaction, i.e., a synchronized behavior, of both worlds have been defined.

It is certainly true that the H-L-PN-based coordination model designed in the last chapter represents correctly the behavior of the flexible production system, and it could be used as basis for performing the addressed control functions. Nevertheless, this model is not sufficiently powerful to do both, to represent the necessary control hierarchy illustrated in section 2.3.1 and to communicate with the real production environment.

Principally, a new set of specifications which must be fulfilled during the design phase of the H-L-PN model and an extended modeling methodology needed to enable the transition from the coordination model to a detailed and functional H-L-PN representation of the components of the DECS.

Firstly, this transition should preserve all the specifications of the production system captured and validated by the coordination model, it has to incorporate new specifications related to control objectives, and it has to be able to allow a synchronized evolution of both, the real production world and the H-L-PN-based DECS.

Secondly, the performed extensions in the H-L-PN coordination model have to cover all new specifications related to the exchange of signals and information between the components of the distributed control architecture, i.e., logic control, monitoring, dispatching, etc., and controlled production environment.

Finally, the conceived H-L-PN-based DECS has to be able to ensure that all the production orders coming from the upper level of the control hierarchy will correctly be converted into control sequences, to detect and recognize the state of the system, and to achieve proper sequencing and synchronization of events in the coordination control level.

Faced with the growing complexity of the flexible production systems, it would also be convenient to hide the technical details of the (real) devices in the control system. From the point of view of the DECS, only addressing of sensors and actuators present as attributes in the H-L-PN model of the controllers would be recommended.

In order to bridge the gaps described above and taking into account the results obtained from chapter 3, the purposes here are:

- to provide an easy-to-understand view of the manufacturing equipment by means of a transformation of the real components of the production world into a kind of virtual technical description (see section 4.1.1), and

- to present an approach to extend the description capacity of the H-L-PN coordination model with complementary attributes and elements for describing and implementing the main functions of some of the highlighted components of the hierarchical and distributed DECS architecture described in Chapter 2 (see section 4.1.2).

4.1.1 Flexible Production Environment with Process Interface

For the purpose of this work, the process interface of a flexible production environment and the flexible production environment self are considered as an integrated entity, e.g., a logic component of the production environment, situated at the bottom level of the hierarchical control architecture.

As illustrated in Fig. 25, the process interface acts as local port structure of the system and is responsible for the information and signal exchange between the production environment and the highest components of the hierarchical control system. From the control and monitoring point of view, the process interface can be considered as the process itself, i.e., there should be no direct access to the drivers of technical devices, such as I/O-ports or object identification systems, in the following referred to as *ident* system. Rather there is only the possibility to access a kind of virtual technical component. The process interface has to hide all unnecessary details in order to provide an easy-to-understand view of the manufacturing equipment /98/.

Detailed discussion about this logic component is beyond the scope of this work. For more information consult a description of an architecture of a process interface based on both, Profibus-DP (Digital Periphery) technology and Dynamic Data Exchange (DDE) based information processing.

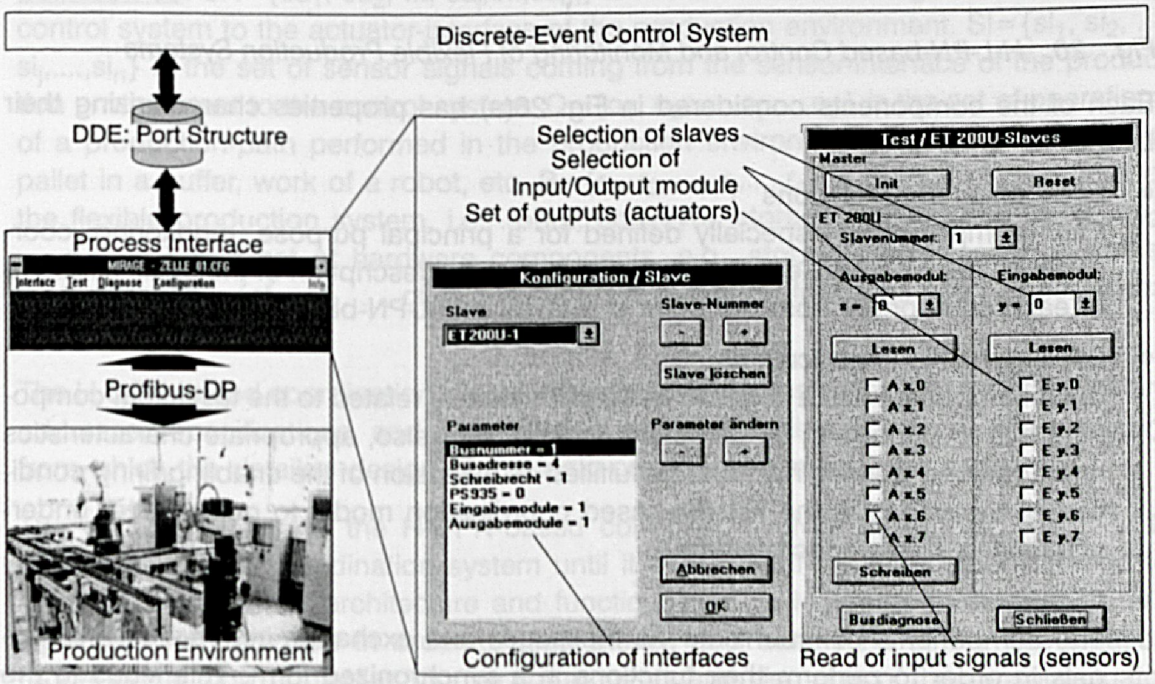


Fig. 25: Flexible Production Environment with Process Interface

4.1.2 H-L-PN-Based Coordination System of the Production Environment

Fig. 26(a) shows in this chapter the part of the DECS structure of Fig. 4 formally and technically designed by using High-Level Petri net as specification tool.

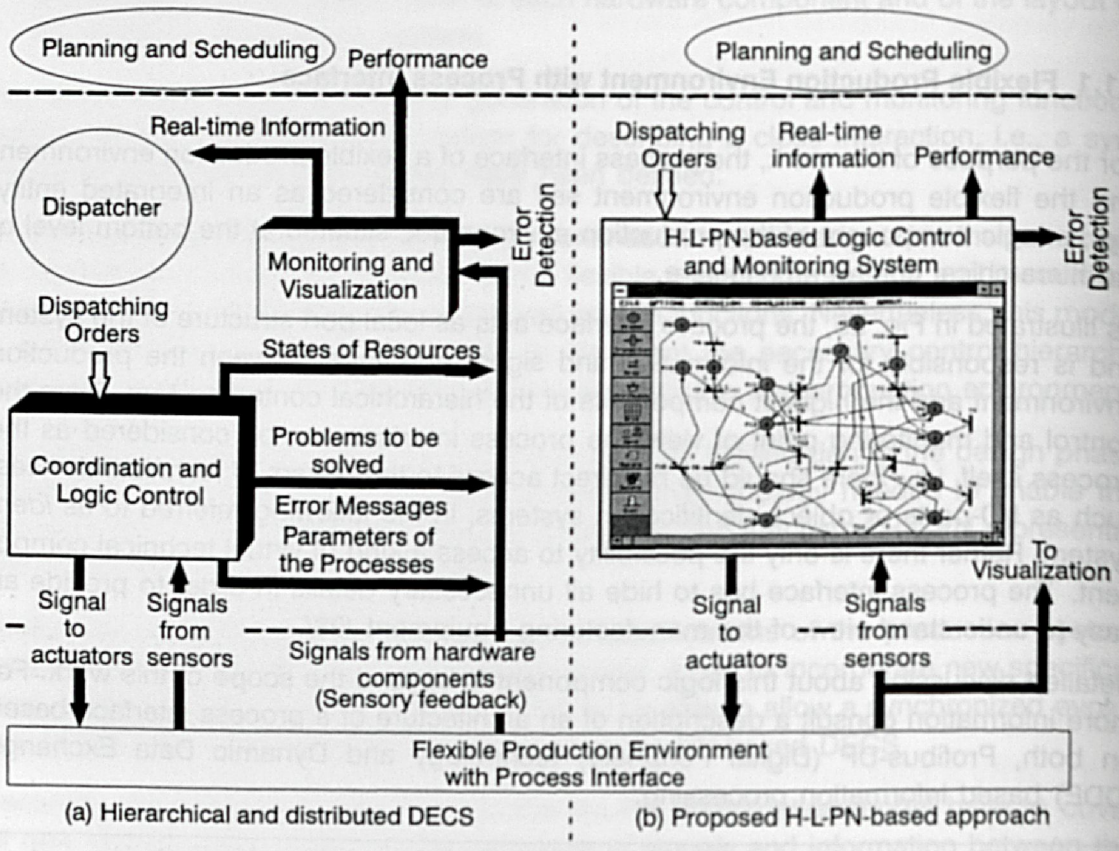


Fig. 26: H-L-PN-based Control and Monitoring of Flexible Production Systems

Each of the components considered in Fig. 26(a) has properties characterizing their behavior:

- Local static specifications
  Each component is especially defined for a principal purpose, monitoring, coordination, logic control, etc. This leads to a specific description of the static structure of each component from the point of view of a H-L-PN-based specification.
- Local dynamic specifications
  Each component has a proper dynamic completely related to the tasks that component has to perform in the structure. In this case, also, appropriate characteristics and additional constraints must be fulfilled as extension of the enabling/firing condition of transitions in the H-L-PN-based coordination model to get a better understanding of these specifications.
- Local port structures
  The components communicate via messages and exchange information and signals in order to perform their functions in a synchronized form. This leads to the definition of new parameters and attributes in the H-L-PN coordination model that

allows the communication and exchange of signals between its elements (e.g., places, transitions, colored markings), the real production environment and the other components of the DECS, i.e., dispatcher, scheduler, planning level, etc.

4.2 Integration of Coordination Functions and Logic Control Functions using H-L-PN-Based Specifications

Material-flow specifications are implemented and manipulated in the production environment by means of production orders, coming out from the upper levels of the hierarchical discrete event control system. However, to ensure that all conversions of the production orders will correctly be done, the operation of the production environment must be controlled by a special unit, able to detect and recognize the state of the system, and to achieve proper sequencing and synchronization of events in the coordination control level. This is a logic control unit which is responsible for the supervision and interaction between physical components of the production environment and the components of the control system /40/, /115/.

Inspired by the description of the layout of the production system and the set of operations to be performed in it, mapping between the hardware specifications of the system, has to be initially done, i.e., resources and components on one hand, and the set of basic tasks to be carried out in each component on the other hand, . The results of this phase can be formally specified as follows:

Definition 35: $SA=\{sa_1, sa_2, ..., sa_j,....,sa_m\}$ is the set of actuator-signals sent from the control system to the actuator-interface of the production environment. $SI=\{si_1, si_2, ..., si_j,....,si_n\}$ is the set of sensor signals coming from the sensor-interface of the production environment to the control system. $O=\{o_1, o_2, ..., o_j,....,o_p\}$ is the set of operations of a production-path performed in the production environment, e.g., movement of a pallet in a buffer, work of a robot, etc. $R=\{r_1, r_2, ..., r_j,....,r_s\}$ is the set of resources of the flexible production system, i.e., robot, conveyor, storage system, etc. $C=\{c_1, c_2, ..., c_j,....,c_h\}$ is a set of hardware components, e.g., stopper, identification systems, motor, etc. $CF=\{cf_1, cf_2, ..., cf_i,...., cf_k]$ is a set of control functions such as *set, reset, read, write*, etc.

The H-L-PN-based coordination model itself contains a substantial amount of the above addressed specifications, and it constitutes now the starting point, i.e., the skeleton, from which the detailed design of the logic control system can proceed.

The detailed design of the H-L-PN-based controller involves developing the H-L-PN description of the coordination system until it contains all the information needed to describe the required architecture and functionality of a logic control system. It is a significant improvement in order to map places and transitions of the coordination model of the logic controller into the components and elements of the sensor/actuator interface of the controlled system.

As final result, the H-L-PN-based model of the logic control system will be made up of two main components hierarchically organized: the coordination control system and a local/logic controller embedded in the first structure.

4.2.1 Extension of the Structure of H-L-PN for Modeling Logic Control Functions

In order to add new possibilities for implementation of logic control architectures using High-Level Petri Nets, additional constraints must be fulfilled as extension of the enabling/firing condition of transitions in the coordination model.

New Considerations about the Classical Guard Form

First modification can be done on the original definition of the net guards. The purpose here is to define additional constraints related to the occurrence-colors (occurrence-modes) of a transition.

Definition 36: The form and structure of the guards $G\&_j(t)$ must include the following set of mathematical operations {"=", "≦", "≧", "<",">"}.
$\forall\, t \in T \Rightarrow [Type([G\&_j(t)]) = Boolean \wedge Type(Variable([G\&_j(t)])) \subseteq C + N]$
The new form of the guards allows to compare components of the complex color domains (e.g., cartesian product color domain) with each other and with integers. For example, if t2 ∈ T:

$\forall\ c_{tk} \in C(t2)$: $G\&_1(t2)=(proj(4).succ(4)=5)$; $G\&_2(t2)=(proj(3)>2)$; $G\&_3(t2)=(\neg c_{tk}) \Rightarrow G(t2) = G\&_1(t2) \vee G\&_2(t2) \vee G\&_3(t2)$

Augmentation of the Transition Enabling/Firing Condition using Timers

Definition 37: For some transitions $t \in T$ in the H-L-PN logic control model, a Boolean function of a non-negative real variable can be defined as *Timer-Guard [TG(t)]*:

$\forall\, t \in T$: $[Type([TG(t)]) = Boolean \wedge Type(Variable([TG(t)])) \subseteq \mathbb{R}^+]$

Example of a Delay-Guard is:

$[TG(t)]=k$, whereby $k \in \mathbb{R}^+$

Two possible forms of using this guard are proposed below.

1) The *Timer-Guard* is used as a *delay* of the firing condition of transition.

1) The *Timer-Guard* is used for implementing a *watchdog mechanism* to detect wrong situations during the evolution of the H-L-PN as logic controller (see chapter 5, section 5.3).

Exchange of Physical Signals between H-L-PN-Based Logic Controller and Production Environment

The evolution of a flexible production system and the information- and signal-exchange between production system and discrete-event control depend continually on the state of the system. In order to implement discrete-event control system, it is also very important to take into account a set of signals coming from the sensor-interface (actual

state of the system) and a set of signals to be sent to the actuator-interface of the controlled system (actions to be performed in order to change a state).

New Boolean functions (guard related to sensor/actuator signals) can be defined.

Definition 38: For some transitions $t \in T$ in the H-L-PN model (coordination control and/or logic control model) a Boolean function of *AND/OR/NOT* logical operations of Boolean variables related to the actuator- and sensor-signals can be defined as *Sensor-Guard [SG(t)]*:

$\forall\, t \in T$: $[Type([SG(t)]) = Boolean \wedge Type(Variable([SG(t)])) \subseteq SA \cup SI]$

An example of *Sensor-Guard* is:

$\forall i,j,k,r,x \in N$,

$[SG(t)] = [si_i \wedge (\neg sa_k) \wedge si_j \wedge ...] \vee [sa_r (\neg si_x) \wedge ...]$

where $\neg si_x$ is the complement of the Boolean variable si_x

$\neg si_x = 1$ exactly when $si_x = 0$

Definition 39: For some transitions $t \in T$ in the H-L-PN model (only logic control model) a function related to the resources, hardware components and to the control functions, which are to be enforced in the production environment, can be defined as *Action-Guard [AG(t)]*:

An example of *Action-Guard* is:

$\forall \quad r_i, c_j, cf_k \Rightarrow [AG(t)]:[r_i.c_j = cf_k]$

So, it is necessary to have the hardware description of the system under study for defining Action-Guard.

Solving Conflicts during the Evolution of a H-L-PN-Based Controller

Despite the enhanced flexibility due to the use of High-Level Petri Nets for modelling discrete-event systems, in a discrete-event control structure based on this tool, a set of conflicts during the evolution of the net has to be solved /71/, /115/. This problem is modeled whether by means of a set of parallel enabled transitions, or a set of occurrence-colors of the same transition in the coordination model, which compete for a common resource /17/. In order to avoid a static solution for this problem as described for example in /22/, a real-time decision system and a planing tool are needed for the support of the evolution of H-L-PN-based logic controllers (see Fig. 4).

Below is defined a new guard related to the transitions of the net. By means of this guard, data and information exchange can be performed between the real-time decision support system and the coordination control level (see next chapter).

Definition 40: For some $t \in T$ in the coordination model and for each $G\&_i(t)$ (definition 36) which occurrence-colors are in conflict with the occurrence-colors of another $G\&_j(t)$, the real-time decision system must define a value of a Boolean decision-variable (e.g., $bdvt_i$, $bdvt_j$). Then, a *Dispatcher-Guard [DF(t)]* is defined as mapping from transitions into expressions of type Boolean, i.e., [Type([*DF(t)*])=Boolean].

$[DF(t)] = \cup_{i \in [1:n]} [bdv_i \wedge G\&_i(t)] =$

$= [bdv_1 \wedge G\&_1(t)] \vee [bdv_2 \wedge G\&_2(t)] \vee ... \vee [bdv_n \wedge G\&_n(t)]$

Corollary 1: For some $t \in T$ in the coordination model and the set of occurrence-colors in conflict $\{G\&_i(t), G\&_j(t), G\&_k(t)\}$ (definition 36), the real-time decision system gives value true to one, and only one, Boolean decision-variable of the set $\{bdv_i, bdv_j, bdv_k\}$.

4.2.2 Modeling

In order to achieve a formal specification of logic control structures using Petri nets or color Petri nets, top-down methodology is now applied so that some elements of the H-L-PN-based coordination model are refined step by step in order to include enough system operation details for the purpose of hardware implementation. For example, the basic transport operations of a modeled production-path are further refined using basic tasks, such as *move*, *place pickup*, *place put in buffer*, *get from buffer*, *stopper out*, *cross transport in*, etc.

This work proposes a stepwise refinement of the the transitions of the net or their firing-mode (occurrence-colors) that model physical operations in a H-L-PN coordination model. Top-down synthesis proceeds by assigning an underlying sub-net (PN/H-L-PN) to some of the transitions, where the new obtained set of Petri net elements (e.g., transitions and places) model technical actions, the processing of real signals to be exchanged between the discrete-event control model and the sensor/actuator interface of the production environment, and the exchange of information among levels of the hierarchical control structure.

The association between elements of a sub-net and basic tasks performed in the manufacturing environment enhances the modeling power by providing a discrete-event control structure which evolves synchronizing with the first one. This synchronization determines the movement of tokens in the H-L-PN-based model and, at the same time, the evolution of the controlled system from one state to the following one. The goal of the refinement method is to establish a set of rules for combining elements of the net that preserve the number and direction of flow of tokens in the original coordination net. This must be the goal, since creating or destroying tokens will alter the liveness and boundedness properties of the net /101/.

The functioning of such a H-L-PN-based logic controller has to fulfill the following behavioral specifications:

- The logic controller unit receives signals input coming out from the production environment. Using this information (actual state of the manufacturing environment) and the actual marking of the net, the coordination model plays the "token-game" of the net.

Explicit specifications of the signals, which are to be exchanged, between production environment and logic control system have been incorporated into some dedicated extensions of the original Petri net interpretation /11/, /26/, /54/, /112/. Input signals affect changes in the state of the system, and are strictly bound to events, and consequently to transitions. Several signals may form a logic function, describing a precondition for an event. Such a function is incorporated in the model by means of the new guards defined in the last section.

- The control unit should be able to assert control enforcement (output signals or instructions) which have to be sent to the controlled system both, when the system stays in a particular state and when a stated event takes places. This means that outputs of the controller can be associated with both, places and transitions. Outputs which are associated with places are Moore-type outputs /4/. They depend only on the local states of the system, and are asserted whenever the associated places have tokens. Outputs associated with transitions depend, not only on the state of the system, but also on the inputs used in the guards associated with the transition. Therefore, this kind of outputs of the controller are Mealy-type and they are asserted whenever the associated transitions are fired /4/.

Note: Only Mealy-type of outputs will be considered in this work.

The refinement method, applied to the transitions of the coordination model that model physical operations, is carried out taking into account the following set of rules:

Rule 1: The sub-Petri nets are a kind of interpreted and synchronized Petri nets /11/, /102/. The evolution of these nets is performed in a synchronized manner with respect to some external signals coming from the manufacturing environment (see section 2.5.3).

Rule 2: For a refined transition in the coordination model a sub-Petri net is defined for each firing-mode or any combination of them.

Rule 3: Each sub-Petri net has a "start-transition / input-transition (without pre-condition places)" and it has an "end-transition / output-transition (without post-condition places)".

Rule 4: The arcs of a sub-Petri net have attached some of the functions previously declared, i.e., functions defined in the coordination model. These functions must be evaluated with respect to the enabled occurrence-color (enabled firing-mode) of the refined transition.

Rule 5: With the transitions of a sub-Petri net can be attached the same types of guards above defined. The purpose of these guards is also to define additional constraints, which must be fulfilled before the transitions are enabled.

Rule 6: Each sub-net has one monitor place. This place will be marked with an uncolored token as initial marking which will be consumed by the start-transition, when it fires. After firing the end-transition, the monitor place recovers its marking.

Rule 7: Technical actions and tasks performed in the manufacturing environment, e.g., robot picks a part, stopper up, etc., are associated to the marking of the places of the sub-nets. The control enforcement, e.g., signals to be sent from the control system to the actuator-interface for performing these actions, is generated from transitions of the sub-nets which pre-condition places become the corresponding tokens. As an example of these possible actions can be considered: send a signal "start-program" to a robot, call a routine for the identification of parts to be processed, etc.

Rule 8: Physical signals (e.g., sensor signals) coming into the sub-Petri net are associated to guards of transitions. As an example of these inputs from the production

environment can be considered: "finish-signal" from a robot-program, part has been identified, pallet reached position, etc.

It is very important to point out that an operation modeled by a transition in the coordination level can be decomposed into a number of elementary control tasks associated with a new set of transitions. Each transition, resulting from the refinement process, implements a sequence of conditioned decisions, resulting in a sequence of control actions, related to one color or tuple of colors of the set Ω^*. If two or more transitions (e.g., occurrence-colors) are concurrently enabled and they are also in conflict, it is extremely distinctive feature that one, and only one, transition (e.g., occurrence-color) can be fired at a time. Because of physical restrictions, i.e., the utilization of a shared resource, only one action can be performed at a time. Normally, the actions performed by refined transitions which are not in conflict can run parallel and interact with each other, giving rise to a concurrent process environment.

The incorporation of a sub-net as a part of the H-L-PN-based coordination model is performed as depicted in Fig. 27.

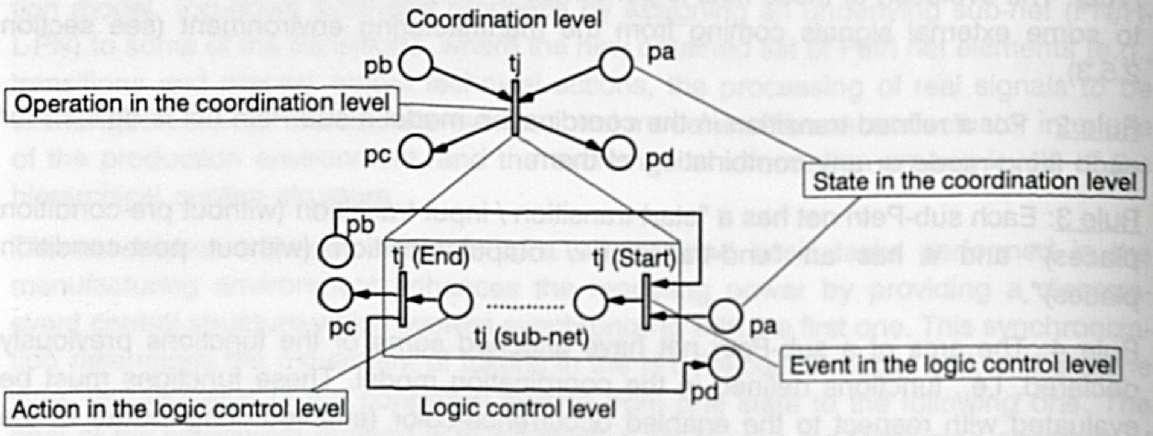


Fig. 27: Refinement of a Transition in the H-L-PN-based Coordination Model

Remark: The top-down method enhances the modeling power by providing the following main characteristic: *the flow of tokens into and out of the sub-net must be the same as in the portion of the original H-L-PN-based coordination model that has been refined.*

Fig. 28 and 29 show the main characteristics of the evolution of a H-L-PN-based logic system and the form of interaction between manufacturing environment and the elements of a H-L-PN-based discrete-event controller, obtained after the application of the above detailed refinement method.

In order to show an application, Fig. 30 presents the refinement of the transition *t2* of Fig. 18 which models the operation "robot 1 loads the work position of robot 2 with a part type 1 from the input position".

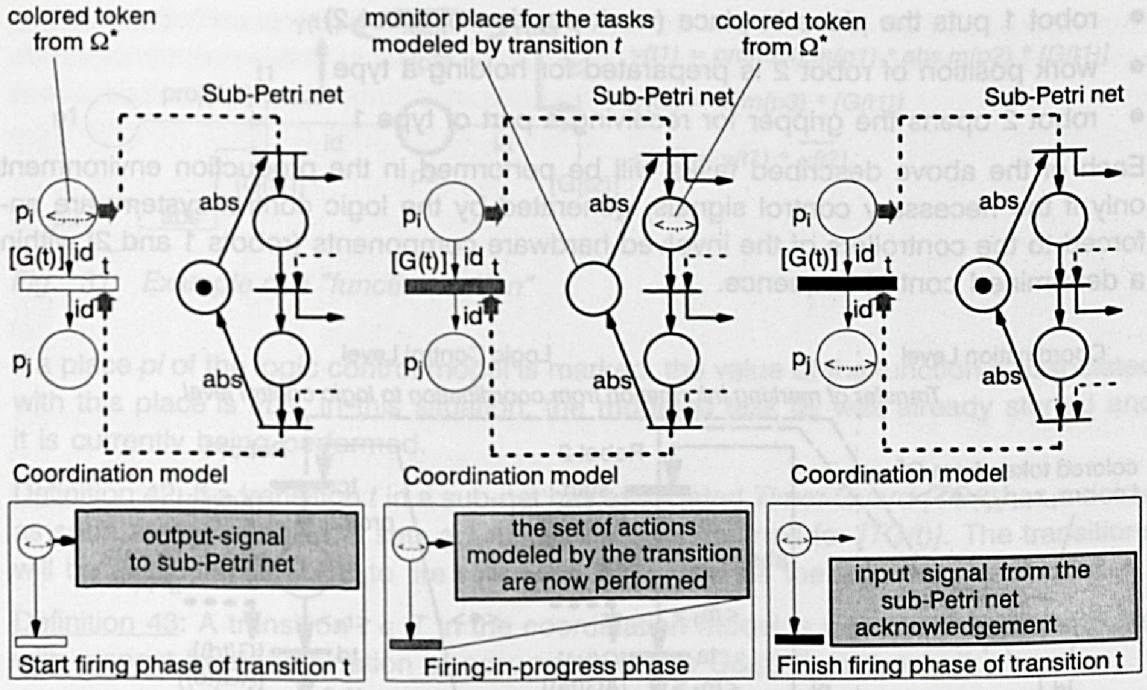


Fig. 28: Token-Game performed in a H-L-PN-based Logic Control System

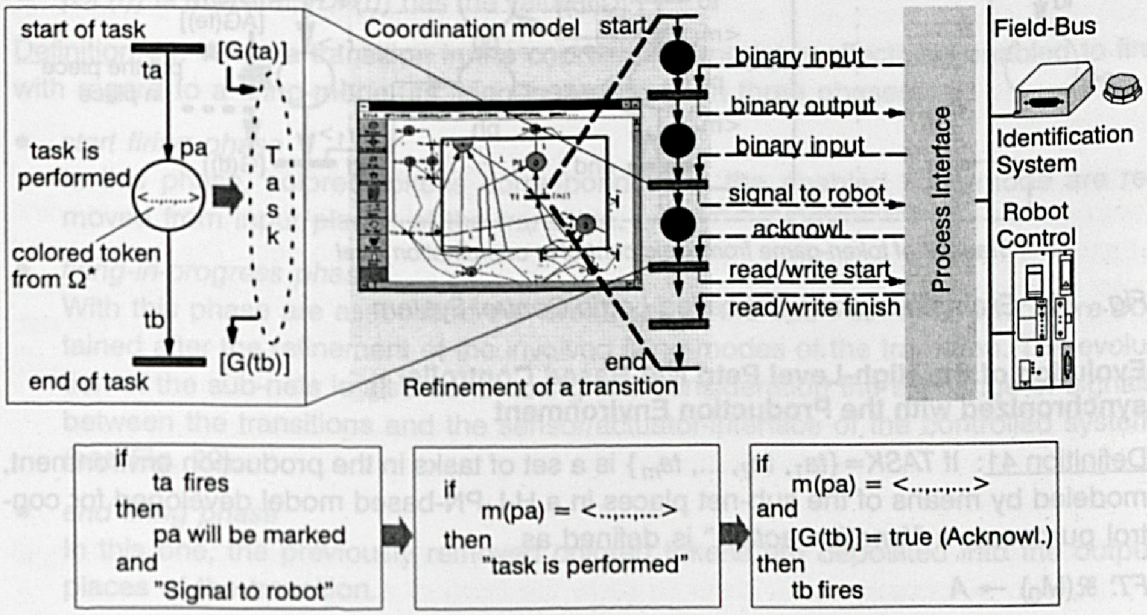


Fig. 29: H-L-PN-based Discrete-Event Controller / Coordination + Logic Control

This operation modeled in the coordination level can be decomposed into the following set of tasks (technical actions) to be done at the logic control level:

- robot 1 picks up the manipulation tool (preparation)
- robot 1 picks up a piece of type 1 and reaches the work position of robot 2

- robot 1 puts the piece in place (work position of robot 2)
- work position of robot 2 is prepared for holding a type
- robot 2 opens the gripper for receiving a part of type 1

Each of the above described tasks will be performed in the production environment only if the necessary control signals, generated by the logic control system, are enforced to the controllers of the involved hardware components (robots 1 and 2) within a determined control sequence.

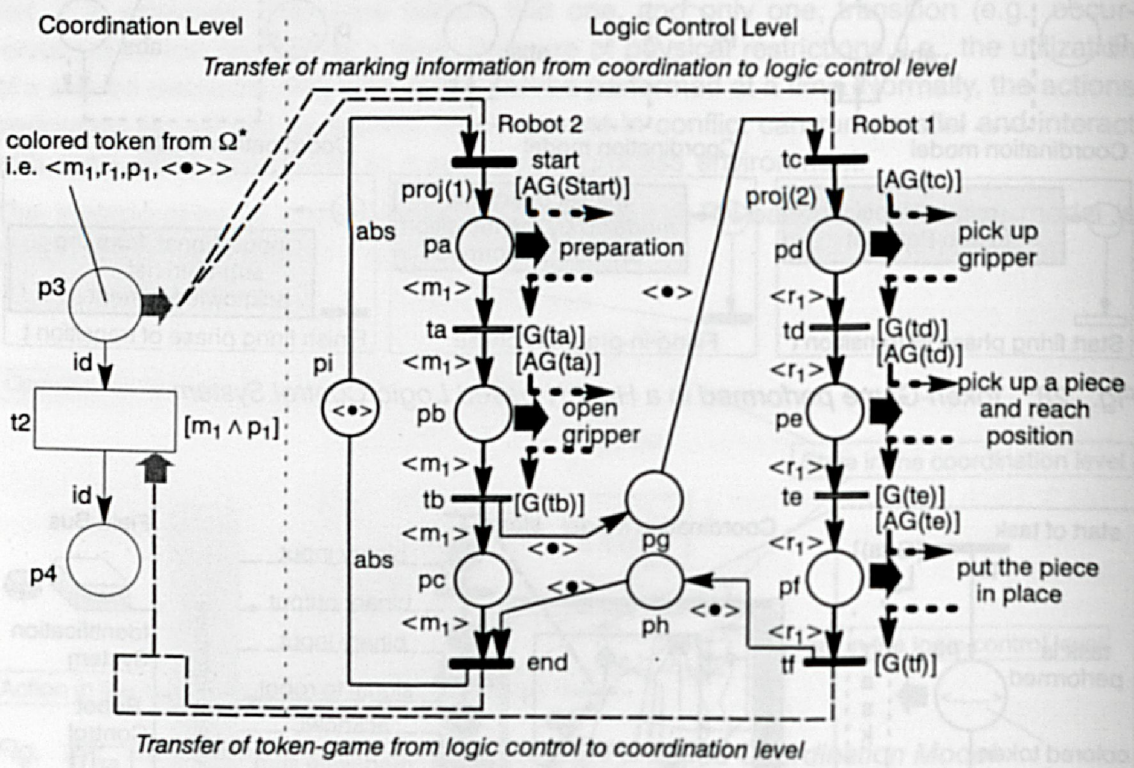


Fig. 30: Example of a H-L-PN-based Logic Control System

Evolution of the High-Level Petri Net-Based Controller synchronized with the Production Environment

Definition 41: If $TASK=\{ta_1, ta_2, ..., ta_m\}$ is a set of tasks in the production environment, modeled by means of the sub-net places in a H-L-PN-based model developed for control purposes, a "*function action*" is defined as

FT: $\Re(M_0) \rightarrow A$

and it can also be seen as a functional vector of Boolean functions

$FT = [\tau_1(m(p1,c_{p1})), \tau_2(m(p2,c_{p2})), \tau_3(m(p3,c_{p3})),..., \tau_n(m(pn,c_{pn}))]$

Note: By considering the evolution rule of an OCPN and the definition of "occurrence-mode functions" (see section 3.3.2) the function FT can be expressed as linear combination of Γ functions.

Fig. 31 shows an application of the above defined function.

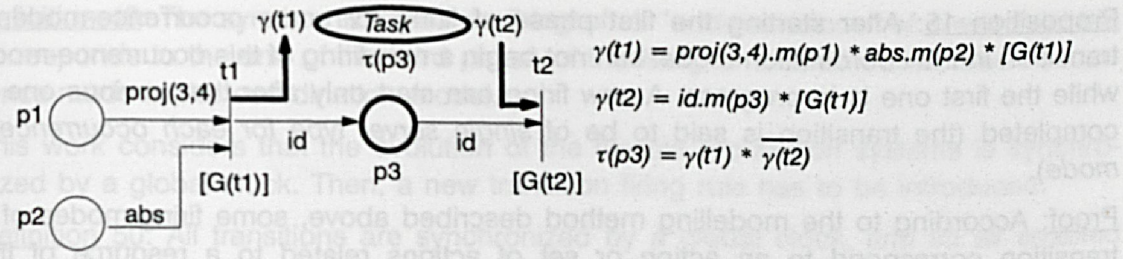


Fig. 31: Example of a "function action"

If a place *pi* of the logic control model is marked, the value of the function τ_i associated with this place is *true*. In this situation, the modeled task ta_i was already started and it is currently being performed.

Definition 42: If a transition *t* in a sub-net has associated *Timer-Guard [TG(t)]* as a delay, as soon as it is enabled, it sets a timer at the value defined for *[TG(t)]*. The transitions will be effectively enabled to fire when the timer reaches the value *0*.

Definition 43: A transition $t \in T$ in the coordination model is effectively enabled to fire with respect to a combination of occurrence-colors $G\&_i(t)$ if, and only if:

- *t* is marking-enabling and the guard $[G\&_i(t)]$ is true,
- *[SG(t)]* is true (i.e., *[SG(t)]* has the value 1) and
- *[DF(t)]* is true (i.e., *[DF(t)]* has the value 1).

Definition 44: When a transition in the coordination model is effectively enabled to fire with regard to a firing-mode, its firing is performed in three phases:

- *start firing phase*
  In this phase, colored tokens corresponding to the enabled firing-mode are removed from input places of the transition.
- *firing-in-progress phase*
  With this phase are associated the evolutions of the sub-Petri nets, which are obtained after the refinement of the involved firing-modes of the transition. The evolution of the sub-nets is also done taking into consideration the exchange of signals between the transitions and the sensor/actuator-interface of the controlled system (see Fig. 29).
- *end firing phase*
  In this one, the previously removed colored tokens are deposited into the output places of the transition.

Summary: When a transition in the coordination model is effectively enabled (for a certain occurrence-color or combination of occurrence-colors) it *fires*, and it then removes tokens from its input places and adds tokens to its output places according to the definitions presented in chapter 2. The number of removed/added tokens and the colors of these tokens are determined by the value of the corresponding arc functions (evaluated with respect to the occurrence-color or combination of occurrence-colors in question).

Proposition 15: After starting the first phase of firing for some occurrence-mode, a transition in the coordination model can not begin a new firing of this occurrence-mode while the first one is in progress. A new firing can start only after the previous one is completed (the transition is said to be of *single server type for each occurrence–mode*).

Proof: According to the modelling method described above, some firing-modes of a transition correspond to an action or set of actions related to a resource of the manufacturing environment (i.e., read a sensor signal and send a signal to an actuator, pick-operation, transport of a pallet, etc.). Each of this actions is unique and can not be repeated before the first one is over (physical restriction).□

Proposition 16: Each modelled operation can be fulfilled in the manufacturing environment only if the corresponding transition (or occurrence-color) is not desabled by the firing of another one in the H-L-PN coordination model. The H-L-PN model must be persistent.

Proof: As stated in section 2.5.4, a Petri net is said to be persistent if, for any two enabled transitions, the firing of one transition will not disable the other ones. A transition in a persistent model, once it is enabled, will stay enabled until it fires according to definition 44.

This definition can be extended to two or more occurrence-colors of a transition in the H-L-PN-based coordination model.□

Definition 45: As soon as a refined transition in the coordination model is effectively enabled and begins the first phase of its firing (definition 44) , the "start-transitions" of the sub-nets associated with the enabled firing-modes are "atomic-fired (firing in one phase)" /9/.

Definition 46: The colored tokens, associated with the enabled occurrence-modes and removed from input places of a transition in the coordination model during the first phase (definition 44), are distributed into the corresponding sub-nets. The number and color of tokens, which evolve in a sub-Petri net, are defined by the enabled firing-mode in the coordination model and the function associated to the arcs of the sub-Petri net.

Note: There are two cases related to the enabling-condition and firing of a transition in a sub-Petri net:

- Firing with an autonomous behavior: the firing is independent from the state of the environment.
- Firing with a non-autonomous behavior: all the conditions presented in definition 43 have to be considered in order to allow the firing of a transition.

Definition 47: As soon as a transition in a sub-net is effectively enabled, with regard to an occurrence-color, it is immediately "atomic-fired (firing in one phase)" /9/.

Definition 48: The firing of a transition in a sub-net produces two concurrent effects:

- token-game of the net, and
- control enforcement via the downloading and execution of the appropriate control functions defined by the *Action-Guard [AG(t)]* associated with the transition.

Definition 49: The synchronized atomic firing of all "end-transitions" of a sub-Petri net corresponds to the end of the third phase of the firing of the refined transition occurrence-mode in the coordination model.

This work considers that the evolution of the flexible production systems is synchronized by a global clock. Then, a new transition firing rule has to be introduced.

Definition 50: All transitions are synchronized by a global clock, and so all enabled transitions (i.e., enabled occurrence-modes) fire simultaneously, and the marking of the net is updated only once per clock cycle. A structure like that of the logic controller is said to be modeled by a synchronous Petri net /102/.

Conclusion

The static structure and the evolution of a H-L-PN-based model of the coordination and logic control system of a flexible production system allows formal classification of this model as "Synchronized High-Level Petri Net", as defined in chapter 2.

At this point is very important to recall the following main issues:

- Control enforcements are generated by the H-L-PN-based DECS and messages are sent from the Action-Guards of the controller to the plant.
- Sensor signals, which flow the reverse way, manage the discrete-event evolution of the H-L-PN-based DECS. The messages coming from the production environment are associated to the Sensor-Guards of the model.

These messages allow both, the model and the real system remaining synchronized.

The DECS at the bottom level is composed of two main *intelligent information systems (IIS)* /110/:

- model of the layout of the flexible production system / flexible production cell (coordination model) which composition involves knowledge and intelligence about the static specifications of the resources and the dynamic behavior of the whole system for achieving production objectives
- model of the control logic embedded into the first one with knowledge about the technical, hardware, composition of the system and sufficiently intelligence to accurately generate the necessary control enforcements related with the above considered production objectives.

4.2.3 Validation of Specifications

The validation of the control logic specifications is a very important step in the DECS development process, because failures, generated during the *setting into operation phase* of the control software, delay the time-to-market, raise the costs or may cause severe damages. Only after the functions of the controller are validated and intensively tested, it should be connected to the real production environment.

Let us assume, that the above addressed H-L-PN models were correctly developed, taking into account the addressed set of rules. In the remainder of this section, a two

phase-based validation-test approach is proposed (for more details, please consult the fundamentals presented in section 2.4.3 and the references /18/, /21/, /41/, /84/). The generated logic structure will be firstly, formally validated by means of functional analysis and then an off-line simulation-based approach will be issued for testing the control logic together with a 3D-kinematics model of the flexible production system to be controlled.

Formal Validation

Below is introduced a set of basic requirements for modelling logic control structures with High-Level Petri Nets. Taking into consideration structural properties of the nets, the main goal of this is to guarantee a formally good behavior of the nets.

- R1) The whole set of places of the net and the places resulting from the refinement of transitions (places belonging to a sub-net) must be covered by place invariants, that is, every place must belong to at least one linear invariant of markings.
- R2) The initial marking of places of the coordination model must be correct.
- R3) All places of a sub-Petri net, except for the monitor place, do not have tokens, neither at the initial marking nor after the firing of the "end-transition".
- R4) Taking into account the definitions of "minimal and canonical support (see section 2.5.4), it is principal condition for a good behavior of the nets, where all place-flows of the model must be canonical (each non-null element must be unitary for each color of the involved elements of $\Omega^{*)}$.

Proposition 17: The sub-Petri net obtained after a refinement of firing-modes in the coordination model has so many canonical place-flows as the number of refined firing-modes.

Proof: According to the refinement rules, presented in section 4.2.2 and the basic definitions of H-L-PN, the incidence matrix of a sub-Petri net corresponding to one firing-mode of a transition in the H-L-PN coordination model is a double diagonal matrix, which elements are determined by the refined firing-color. The diagonalization of such a matrix is simple and allows proofing the existence of only one place-flow involving all the places of the sub-net that have relation with the refined firing-mode.□

Corollary 1: The number of place-flows of the OCPN-coordination model is augmented exactly in the number of canonical place-flows of the sub-Petri nets obtained after applied the refinement of firing-modes.

Lemma 2 (Relation between colored tokens and technical actions): During the evolution of the H-L-PN model each place-flow of a sub-net possesses, one and only one, colored token which is responsible for sending signals from the net to the actor-interface of the controlled system.

Proof: Immediate by considering the requirements R1, R2 and R3 together with proposition 15.□

- R5) The transitions of a sub-net must be covered by transition-flows. For each transition and its corresponding firing-modes, at least one transition-flow can be found

that contains it as an element. This condition assures the possibility of cyclic behavior.

Proposition 18: Each sub-Petri net, obtained after the application of the top-down method in the coordination model, has one and only one transition-flow related to each refined firing-mode or combination of them. The cardinality of a transition-flow corresponds to the number of transitions that comprise the sub-Petri net.

Proof: According to the refinement rules presented in section 4.2.2, the transpose of the incidence matrix of a sub-Petri net corresponding to one firing-mode of a transition in the H-L-PN coordination model is a double diagonal matrix, which elements are determined by the refined firing-color. The diagonalization of such a matrix is simple and allows proofing the existence of only one transition-flow involving all the transitions of the sub-net, which have relation with the refined firing-mode. Then, each sub-net of the logic control model is a mono transition-flow net /9/.□

Note: Each operation in the production environment is modeled by means of a transition of the coordination model. The refinement method above addressed provides only a sub-net for this transition. The sequence of actions is unique. These actions are to be generated by the logic controller in order to complete an operation in the production-path. The behavior of the logic control system is totally deterministic and there is not a possibility for conflict situations. This is possible, if and only if, the sub-net has, one and only one, transition-support.

- R6) The sequence of actions performed in order to complete an operation in the production-path corresponds to an equivalent occurrence/firing sequence of the sub-net with firing count vector Θ matching with the unique transition-flow of the net. Then, the unique transition-flow of the sub-net has to be found for each control sequence issued from a transition of the coordination model to the production environment. With this transition-flow a repetitive or cyclic behavior of the sub-net for the proposed logic control structure can be validated.
- R7) The H-L-PN must be live. Then, all the actions associated with the places of the net can become a colored token and the corresponding associated action can be started.

Remark: The last requirement is not easy to proof, because the check of liveness can be replaced by the test of deadlockfreeness /117/.

By taking into account these requirements, the approach presented here provides criteria to ensure that the method for refining High-Level-Petri Net models of logic control structures does not destroy the correctness of criteria for optimal control of the discrete-event coordination system developed in chapter 3.

For the sake of simplicity and without loss of generality, we restrict the attention to the model depicted in Fig. 30. The structural analysis of this H-L-PN allows calculation the following set of place-flows (p-flows)

- *p-flow1: [0 0 0 pd pe pf 0 0 0]* (relative to the robot 1).
- *p-flow2: [pa pb pc 0 0 0 0 0 0]* (relative to the robot 2).

- *p-flow3: [0 0 0 pd pe pf pg ph 0]* (relative to the coordination of both resources)
- *p-flow4: [pa pb pc pd pe pf pg ph pi]* (relative to all involved tasks)

and the following set of transition-flows (t-flows)

- *t-flow1: [Start ta tb tc td te tf End]* (relative to the firing-in-progress phase of transition *t2* of Fig. 18).
  Taking into consideration the functions and guards associated to the arcs and transitions of the net and the proposition 18, this t-flow can be also written
  t-flow1: [(proj(1)+proj(4)) proj(1) (proj(1)+proj(4)) (proj(4)+proj(2)) proj(2) proj(2) (proj(2)+proj(4)) (proj(1)+proj(4))]=
  $=[(<m_1>+<\bullet>)\ <m_1>\ (<m_1>+<\bullet>)\quad (<\bullet>+<r_1>)\ <r_1><r_1>\ (<r_1>+<\bullet>)\ (<m_1>+<\bullet>)]$

From these results, it is possible to ensure that the proposed model correctly represents the given specifications. For example:

- From p-flow1 can be concluded that places *pd, pe* and *pf* can contain maximum one token with the color $<r_1>$ and they are in mutual exclusion states during the evolution of the net (proposition 17 and lemma 2).
- The states in which the robot 2 $<m_1>$ can be found are
  –free before working (marking $<m_1>$ is in place *p3* as part of the tuple $<m_1,r_1,p_1,<\bullet>>$)
  –preparation (marking $<m_1>$ in place *pa*)
  –open gripper (marking $<m_1>$ in place *pb*)
  –wait for acknowledgment "piece in work position" (marking $<m_1>$ in place *pc*)
  –wait for working (marking $<m1>$ is in place *p4* as part of the tuple $<m_1,r_1,p_1,<\bullet>>$)

Structural analysis applied to *t-flow1* allows validating, among others, of the following specifications:

- The enabling-condition and the firing of transition *te* issues the following events:
  – The logic controller sends the signal "robot 1 $<r_1>$ put a piece in work position of robot 2 $<m_1>$" to the actuator-interface of the manufacturing environment by setting the action-guard *[AG(td)]* (action / task is issued).
  – The logic controller reaches a state with the token $<r_1>$ in place *pf*. This state shows that the operation "put the piece in place" is right now performed.
  – The logic controller waits for an acknowledgment from the sensor-interface of the manufacturing environment by testing the value of the sensor-guard *[G(tf)]*.

Note: After these events have been carried out, the component responsible for performing the modeled task (robot 1) works and the sub-net is marked. Two synchronized events are issued, one in the real production system and another in the modeled DECS.

- The enabling-condition and the firing of transition *tb* for the occurrence-color $(<m_1>+<\bullet>)$ are produced after the evaluation of the markings $<m_1>$ of *pb* and the sensor-guard [G(tb)].

- The firing sequence composed of transitions {start, ta, tb, tc, td, te, tf, end} allows the H-L-PN to return to the initial marking (the net is reversible for this sequence of firing of transitions).

Integration of H-L-PN-Based Discrete-Event Platform and Motion-Oriented Simulation

The result of the first approach is a first validated design of the logic control structure that guarantees a correct behavior (e.g., deadlockfree, repetitive evolution, boundedness of logic states, etc.). A complementary method for the validation is to carry out a discrete event simulation that is mainly based on the token-game of the H-L-PN-based controller.

In the following, this validated logic control structure has to be tested on the real production system and optimized in order to fulfill the specifications. However, it is very costly and it might be the case that the real system is not available at this time. Additionally, failures in the control logic could cause damages or endanger lifes. Therefore, a simulation model of the real manufacturing environment based on a motion-oriented, i.e., a 3D-kinematics, platform can be used as the controlled system and also used to validate the control enforcements issued by the H-L-PN-based discrete-event controller.

Only if the test of the control logic with the motion-oriented simulation of the controlled system is successful, the logic controller can be connected with the real production environment.

In order to use a 3D-kinematics model for validation and test logic control software, the model must have the same static and dynamic behavior as the H-L-PN-based discrete-event system and the real manufacturing environment. Only then, a reliable statement on the control logic can be made. This means that all elements in a real manufacturing environment that interact with the discrete-event control system, e.g., actuators and sensors, have to be modelled. As a consequence, also every possible error that can encounters in a real cell, has to come up in the model under the same circumstances.

Fig. 32 shows the main steps, which must be performed for validating and testing the control logic, embedded in a DECS structure using such a simulation-based approach:

- analysis of the specification of the manufacturing environment, i.e., specifications of resources, layout and production environment plus DECS;
- building a CAD-model, building a kinematics model and adding of logical information in the motion-oriented platform;
- validation of specifications and test of the control logic by evaluating the results of the motion-oriented simulation controlled by the H-L-PN-based discrete-event platform.

Some results on validation of the flexible production system's behavior using simulations approaches have been elucited. Little has been done to generalize these results to validation and test of control logic software for such kind of production systems and to help for implementing these simulation-based methods at industrial level.

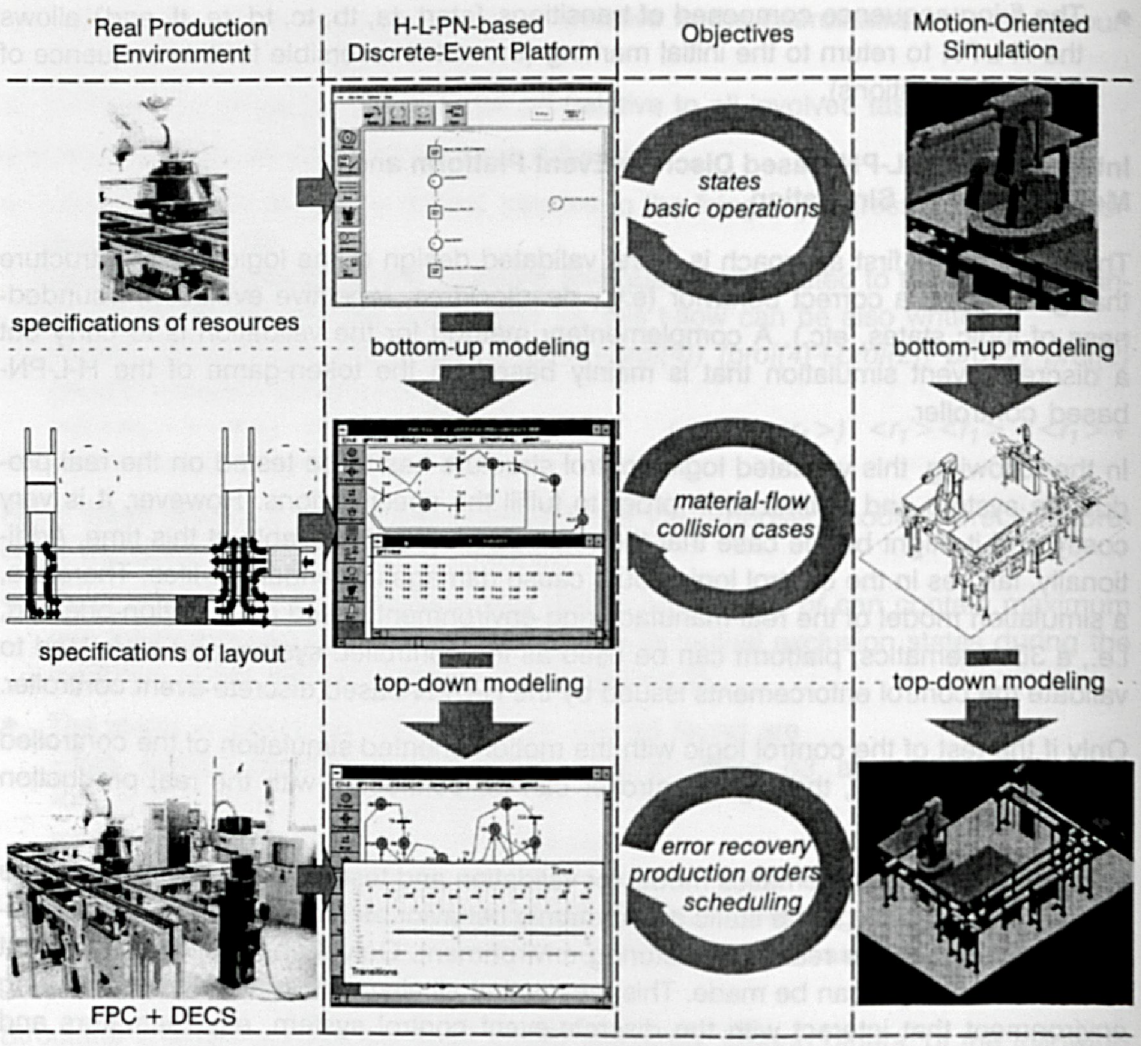


Fig. 32: Discrete-Event and Motion-Oriented Simulation to Validate Control Logic

An off-line simulation approach combining Petri nets in order to satisfy the discrete aspects of a flexible manufacturing system and motion-oriented simulation for the continuous aspects is proposed by /10/. The main goal was to dimension the system under study and to determine process control parameters. However, neither a closer look how the simulation should be performed, nor special aspects dealing with the test of the control logic, were presented.

In order to close these gaps, two possible approaches are here addressed.

The first one consists on connecting the discrete-event platform with the motion-oriented simulation platform using the same process interface described in section 4.1.1. In this case, the process interface is a mapping layer between three different systems: the H-L-PN-based discrete-event controller, the 3D-kinematic platform and the real production environment. Fig. 33 depicts the structure for implementing this approach /84/.

The second method for implementing a simulation-based validation and test of control logic is based on the integration of both, H-L-PN-based discrete-event controller and

motion-oriented simulation system. In this case, only one system is responsible for performing the following main functions:

- validation of the flexible production system's behavior;
- validation of the control logic embedded in the DECS structure associated to the FPS;
- setting the controller into operation.

An approach like this one for the off-line programing of a robot placement system for electronic components is proposed in chapter 6.

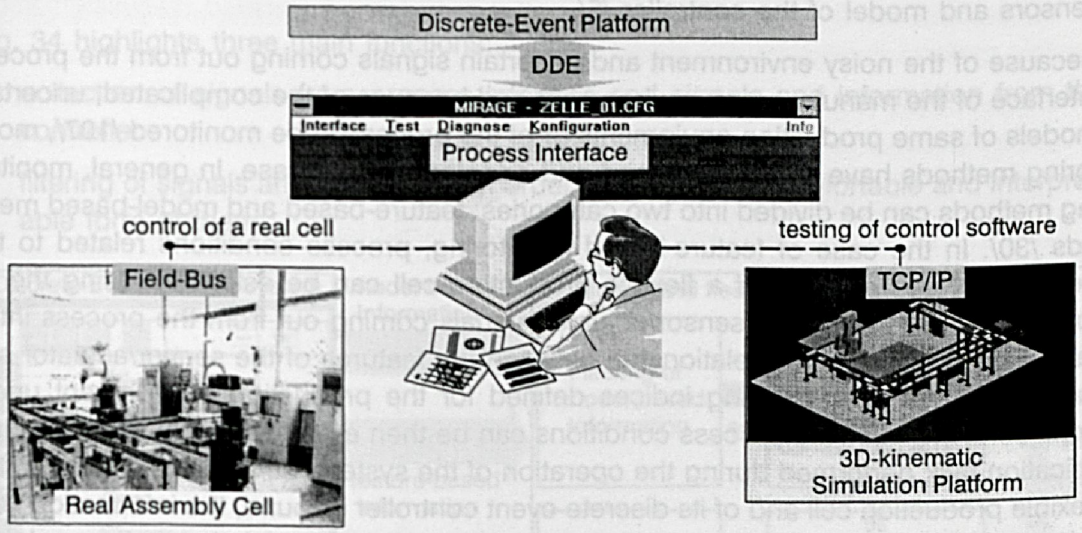


Fig. 33: Connection of Discrete-Event Control and Motion-Oriented Simulation Platform

4.3 Model- and Feature-Based Monitoring

Basic Concepts

High reliability of flexible production systems is a very important factor in gaining a good position on the market. A very useful way to achieve this goal is to learn very well the system and its working conditions (process conditions), and also the parameters of operation of the embedded discrete-event control system itself.

In the changeable operations of flexible production systems, the only form of increasing reliability is to use well adapted monitoring methods. Due to new developments in sensor techniques and signal and information processing systems, as well as deeper knowledge of the production processes and controlled equipment, there are growing possibilities of using reliable, effective and low-cost monitoring systems /107/.

Generally speaking, monitoring a flexible production system requires three successive phases. Firstly, the validation of the specifications of the production system, of its discrete-event controller, and of the implementation (detection of coding bugs). The second step concerns the "on-line sensing" and "information collection" which is performed with taking into account the real-time evolution of the production environment

and of the control system. Finally, information and sensor-signal processing has to be performed in order to obtain a set of indices related to the production environment and its discrete-event control system /96/.

The application of these monitoring phases is only possible if the components of the lower level of the hierarchical DECS are capable to handle enough information about the evolution of the process and the behavior of components of the flexible production environment, which is to be monitored. The resulting monitoring system has to generate the capability for the components of the DECS to communicate with the various devices constituting the cell, and to gather the information provided by the devices' sensors and model of the controller /5/.

Because of the noisy environment and uncertain signals coming out from the process interface of the manufacturing environment and because of the complicated, uncertain models of same production equipments or of the process to be monitored /107/, monitoring methods have to be developed and adapted to each case. In general, monitoring methods can be divided into two categories: feature-based and model-based methods /30/. In the case of feature-based monitoring, process conditions related to the behavior of components of a flexible production cell can be estimated using the information provided by the sensor/actuator signals coming out from the process interface. Taking into account relationship between the features of the sensor/actuator signals and a set of monitoring indices defined for the production environment under consideration, real-time process conditions can be then estimated by means of a classification task performed during the operation of the system /30/. When a model of the flexible production cell and of its discrete-event controller is found, the information contained in it and the evaluation of the parameters of the model during the real-time operation of the system allows to perform monitoring functions. The approach is based on the detection of changes of the parameters of the models and also on the relationship between this parameters and the specifications of the modeled system. Such a model-based monitoring can only be implemented if the flexible production system and its discrete-event controller are correctly modeled and the selected modeling tool has enough capabilities for representing static and dynamic specifications of both systems.

From a monitoring point of view, both integrated H-L-PN-based models (coordination and logic control) can be considered as sources of a lot of information about the process developed in a controlled flexible production cell, e.g., chronology of the operations performed in the production environment imposed by the production plans, and also about the behavior of the discrete-event controller itself.

Both kind of monitoring methods can be applied:

- feature-based monitoring using the information contained in the process interface and also in the sensor- and action-guards of the H-L-PN-based logic control structure, and
- model-based monitoring by means of the evaluation of both parameters of the H-L-PN-based model of the controller, and of the information obtained during the evolution of this model synchronized with the behavior of the hardware components of the controlled flexible production cell.

Fig. 34 depicts a structure for performing these monitoring functions. As shown in Fig. 26, these functions have to be done in a module which works separated from the coordination and logic controllers, but these structures operate in close interaction, i. e., on the same H-L-PN-based model of the production system and its discrete-event controller. Consequently, this structure has the capability to update a set of monitoring indices *mi*, in run time, with perceived information coming out from the process interface and also from the H-L-PN, without modifying the normal behavior of the production environment. A main result of this approach is that real-time model-based and feature-based monitoring of the process are concurrently performed with the control of the process.

Fig. 34 highlights three main functions:

- collection of signals from process interface and signals and information from the controller
- filtering of signals and information in order to get a more comfortable and interpretable form of them.

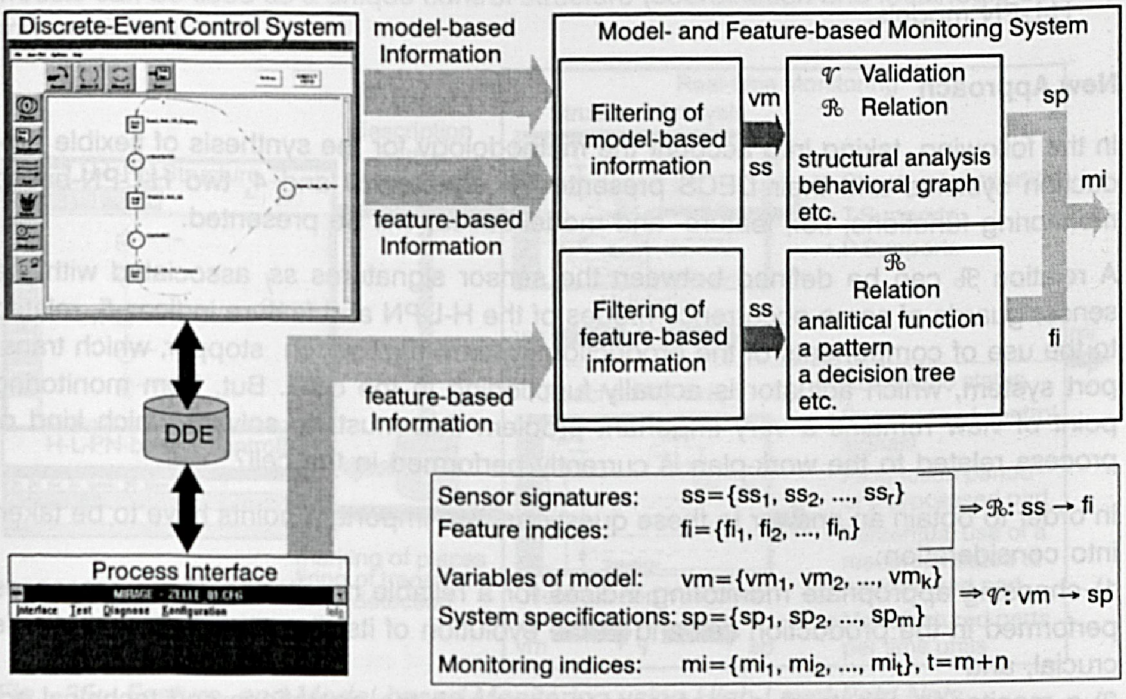


Fig. 34: Structure of a H-L-PN-based Monitoring System

The sensory feedback coming out from process interface and also the information obtained from the model of the discrete-event controller is not always expressed in comprehensive terms /5/. After the collection of signals and information, the second monitoring function (filtering) translates these into two sets which can be easily understood and processed. The first one is a set of model variables $vm = \{vm_1, vm_2, ..., vm_k\}$, such as colored marking of places, occurrence-modes of transitions, guards associated with transitions, incidence matrix of the net, etc. The second one is a set of sensor signatures $ss = \{ss_1, ss_2, ..., ss_r\}$, associated to signals of sensors and actuators of the

process interface, such as a barcode supplied by an identification system associated with pallets or other components of the production environment, etc.

- processing of signals and information to obtain monitoring indices.

The third monitoring function can be carried out in one of three ways depending on the set considered as a source of information and the used monitoring method:

- feature-based monitoring −> processing of the information contained in the set *ss* by means of two phases: "learning" of a relation $\Re$ between the set *ss* and a set of feature indices *fi*, and "classification" of the sensor signatures for obtaining the feature indices *fi* during the evolution of the system. A detailed discussion about this method is beyond the scope of this work /30/;
- model-based monitoring −> processing of the information contained in the set *vm* using analysis techniques of the structure of the H-L-PN-based models
- feature-based monitoring −> processing of the information contained in both sets *ss* and *vm* using analysis techniques of the dynamic (evolution) of the synchronized H-L-PN models.

New Approach

In the following, taking into account the methodology for the synthesis of flexible production systems and their DECS presented in chapters 3 and 4, two H-L-PN-based monitoring functions, i.e., feature- and model-based, will be presented.

A relation $\Re$ can be defined between the sensor signatures ss_j associated with the sensor-guards of some occurrence-modes of the H-L-PN and feature indices fi_k related to the use of components of the production system, e.g., which stopper, which transport system, which actuator is actually functioning in the cell?. But, from monitoring point of view remains a very important problem that must be solved: which kind of process related to the work-plan is currently performed in the cell?

In order to obtain an answer to these questions, two important points have to be taken into consideration:
1) choosing appropriate monitoring indices for a reliable monitoring of the processes performed in the production cell and of the evolution of its discrete-event controller is crucial, and
2) a monitoring construct is necessary to cope with the operations and technical actions performed in the flexible production system and its control structure.

Assuming that the variables of the model have been identified, that the analysis of the structure of the H-L-PN model has been performed and that a set of its properties has been calculated, i.e., transition-supports, place-supports, mutual exclusion conditions between colored marking of places, etc., a validation Υ of the specifications of the system and of its controller can be performed. From monitoring point of view, the same validation technique allows obtaining a set of variables identified as $sp = \{sp_1, sp_2, ..., sp_m\}$ (specifications of the flexible production system and its controller), i.e., material-flow specifications, control sequences, etc.

The real-time synchronized evolution of the H-L-PN token player behaves as an observer of the production environment and its control system /96/. It performs naturally monitoring functions. In this case, an on-line reachability analysis of the net evolution can be seen as a new relation $\mathfrak{R}$ from which a set of new feature indices *fi* can be obtained, e. g., reachable states, production period of a processed part /22/, error detection, etc.

Since both monitoring methods are concurrently performed, the set of monitoring indices *mi* supplied by the proposed monitoring structure can be respectively defined as the combination of both sets, *fi* and *sp*.

Fig. 35 shows a monitoring construction developed at the Institute for Manufacturing Automation and production Systems, which performs a combination of both, model-based and feature-based monitoring methods.

As a matter of fact, the H-L-PN models of the coordination and logic control components of the DECS structure are also used as a source of reliable information about the behavior of the system for building monitoring indices. This means that both integrated models can be used as a unique control structure (coordination and local/logic control) and monitoring tool.

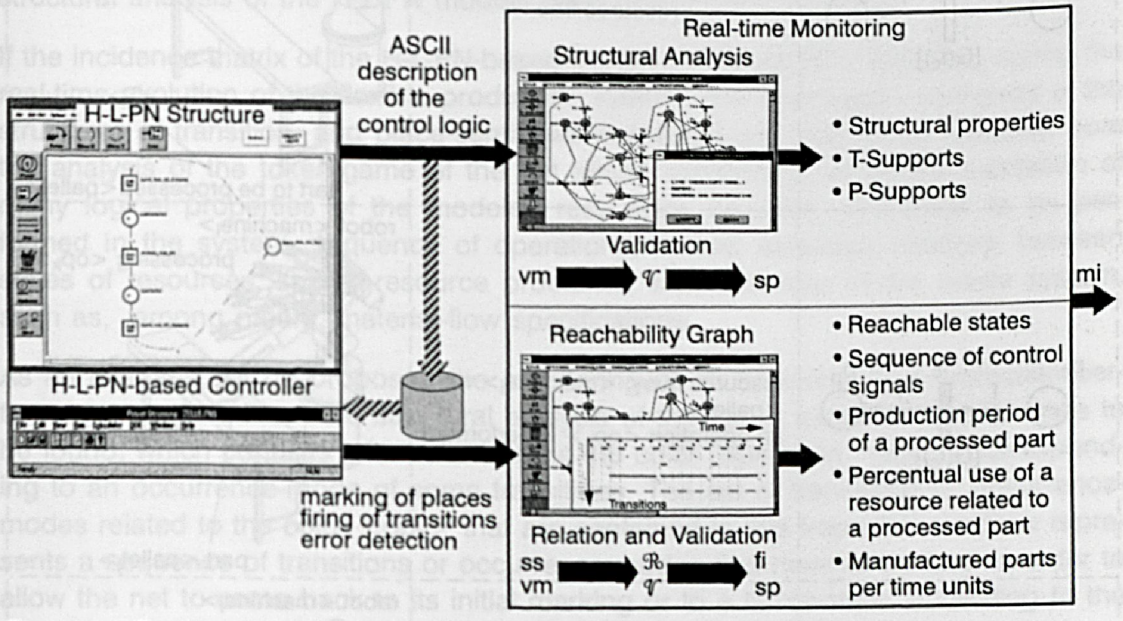


Fig. 35: Feature- and Model-based Monitoring using High-Level-Petri Nets

4.3.1 Model-Based Monitoring using the Real-Time Analysis of the Token-Game of H-L-PN Coordination Control Models

As explained in chapter 3, when the H-L-PN is executed, a colored token in a place of the coordination model will indicate the state of a resource: already free or busy by a part, which part has been loaded in the resource, which task of a work-plan can be performed, etc. The mechanism for performing the token-game of a H-L-PN-based coordination model that was explained in chapter 3 is depicted now in Fig. 36.

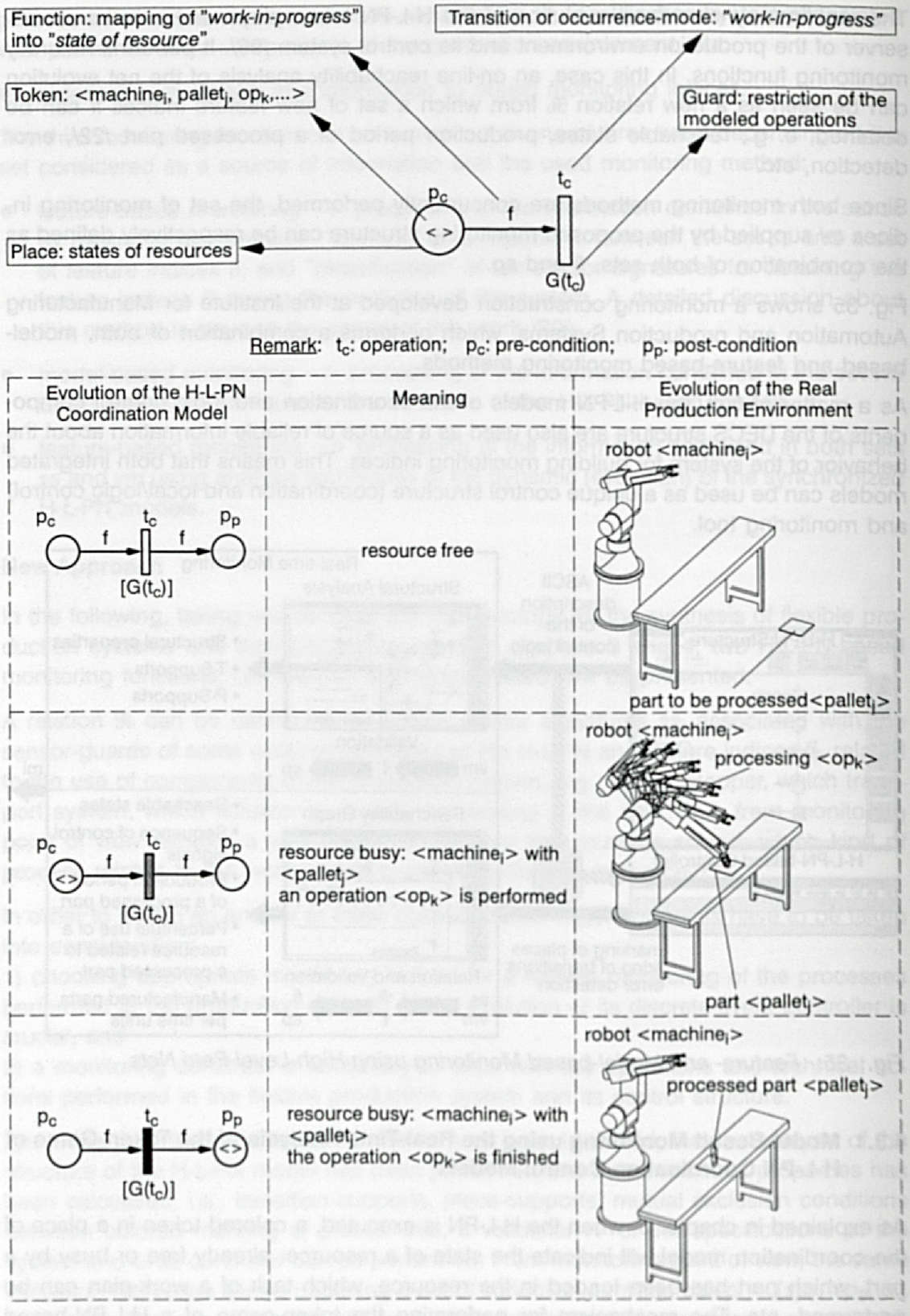


Fig. 36: H-L-PN Model-based Monitoring of the Processes in a FPC

An analysis of Fig. 36 shows that the enabling condition of a transition or of an occurrence-mode of a transition will indicate, for example, which operation <op_k> is already performed on a part <$part_j$> in the resource <$machine_i$>.

In this case, it appears clearly that the monitoring system has an accurate knowledge about the state of each modeled resource of the flexible production cell and also about the operations actually performed in it. The token-game of the H-L-PN coordination model, synchronized with the evolution of the production environment, which acts as a source of process signatures and the real-time analysis of the reachable markings of the net, allows the generation of a lot of feature indices, such as percentual use of a resource related to a processed part, manufactured parts per time units, etc. /22/.

4.3.2 Model-Based Monitoring combining Real-Time Analysis of the Net Structure and the Information obtained from the Token-Game

This kind of monitoring functions is performed by means of the same methods for the structural analysis of the H-L-PN models used during their synthesis.

If the incidence matrix of the H-L-PN-based model of the DECS is analyzed during the real-time evolution of the flexible production system, the information contained in the structures of transition- and place-semiflows of the net and also these obtained from the analysis of the token-game of the net allows performing an on-line validation of many logical properties of the modeled resources: possible work-plans to be performed in the system, sequence of operations, mutual exclusion relations between states of resources, shared-resource problems, etc., and also of the whole system, such as, among others, material-flow specifications.

As an example, let be proposed the monitoring of an operation of a work-plan performed on a part. From the structural analysis of the net, a transition-semiflow has to be found, which contains information about the color "operation <op_k>" corresponding to an occurrence-mode of some transitions. The set of transitions or occurrence-modes related to the color <op_k> that are contained in this transition-semiflow represents a sequence of transitions or occurrence-modes that have to be fired in order to allow the net to come back to its initial marking or to a home-state. According to the modeling method issued in chapter 3, this sequence of transitions or occurrence-modes related to the color <op_k> corresponds to a sequence of technical operations to be performed on a part in the manufacturing environment. When the net evolves and a transition or the occurrence-mode <op_k> fires, this information plus the information contained in the reachability graph (which transitions have been already fired with respect to this color) allows determination about which operations of the work-plan have been already performed in the cell.

Note: The above depicted monitoring mechanism implies that the monitoring system must combine both, the information from the structural analysis and also from the token-game of the H-L-PN model to perform this monitoring function.

4.3.3 Feature- and Model-Based Monitoring of the Hardware Components Behavior in Flexible Production Cells
Information obtained from the Structural- and Behavioral-Analysis of H-L-PN Logic Control Models

It is certainly true that the coordination model addressed in the last section represents correctly the behavior of the flexible production system, but the use of such a model as basis for obtaining information from the control system adds a new set of restrictions which must be fulfilled before the monitoring system is implemented.

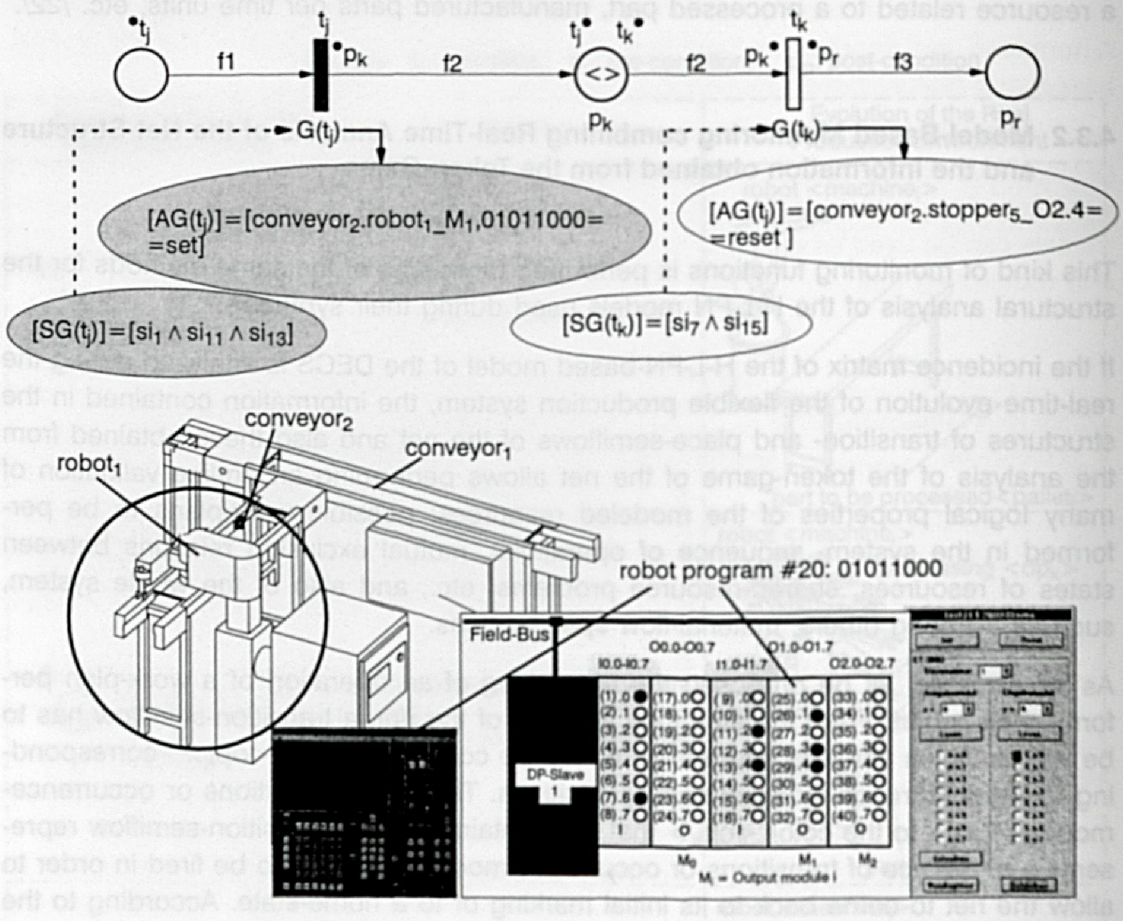


Fig. 37: Monitoring of the Behavior of a FPC and its DECS, State 1

As described in this chapter, the H-L-PN coordination model of the system (e.g., model of the process) is refined and then it constitutes the skeleton of the discrete-event control system. The association between elements of a sub-net and basic tasks performed in the manufacturing environment enhances the modeling power by providing a discrete-event control structure which evolves synchronized with the first one. This synchronization determines the movement of tokens in the sub-nets (logic control model) and, at the same time, the evolution of the controlled system from one state to the following one. Both systems together act as a rich source of new monitoring indices.

When the token-game of the H-L-PN-based model of the discrete-event control system is executed, real-time information about the technical actions performed in the resources of the manufacturing environment can be obtained. Feature- and model-based monitoring functions are naturally implemented in the discrete-event control module. In fact, as pointed out in section 4.2.2, the refinement of transitions or occurrence-modes of the colored Petri net coordination model allows obtaining a new view of the production environment: the flexible production system, from the point of view of the technical components and of the actions to be performed in them, in order to complete the production processes.

Fig. 37 and Fig. 38 show two consecutive states of a part of a flexible assembly cell and the corresponding control enforcements generated by the H-L-PN-based logic controller.

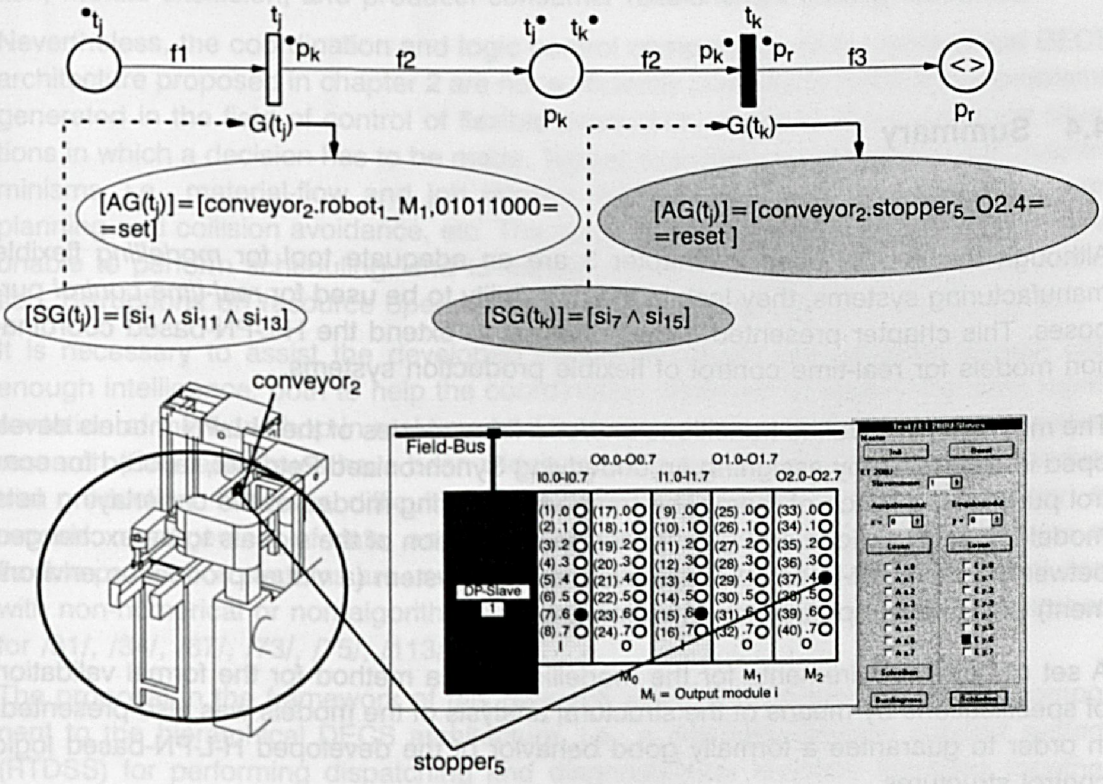


Fig. 38: Monitoring of the Behavior of a FPC and its DECS, State 2

By analyzing of both states of the cell, following considerations about monitoring functions can be derived:

- In H-L-PN-based model of Fig. 37 $•t_j$ is a model-based pre-condition of the action "load robot_program #20".
- In H-L-PN-based model of Fig. 37 the value of the sensor-guard *[SG(tj)]* is a feature-based pre-condition of the action "load robot_program #20".
- t_j fires iff $f1.m(•t_j)=1$ and $[SG(t_j)]=true$. Two concurrently control functions are generated, the marking of $p_k \equiv t_j•$ (logic function) and the control enforcement described

by the action-guard *[AG(t_j)]* (technical function). From monitoring point of view, a model-based monitoring function is performed taking into account the information of the colored token of the place p_k, and a feature-based monitoring function by considering the control signals sent from the H-L-PN-based controller (guard *[AG(t_j)]*) to the process interface.

- $p_k \equiv t_j^\bullet$ remains with the token "<>" as long as the sensor-guard *[SG(t_k)]* is false. This means that the marking $m(p_k)$ performs a model-based monitoring function related with the tasks performed by the $robot_1$ and other informations contained on "<>".
- Fig. 38 depicts all control and monitoring functions related with the operation of a part of the $conveyor_2$, i.e., $stopper_5$.

4.4 Summary

Although the H-L-PN used in chapter 3 are an adequate tool for *modelling* flexible manufacturing systems, they lack in the possibility to be used for *real-time control* purposes. This chapter presented a methodology to extend the H-L-PN-based coordination models for real-time control of flexible production systems.

The main idea is to refine transitions and/or a firing-modes of the H-L-PN models developed in chapter 3, by assigning an underlying Synchronized Petri net, tailored for control purposes, to each of them. The transitions or firing-modes of the underlaying nets model technical actions and give a concise description of the signals to be exchanged between the H-L-PN-based discrete-event control system (a virtual production environment) and the real production environment.

A set of basic requirements for the modelling and a method for the formal validation of specifications by means of the structural analysis of the models was also presented, in order to guarantee a formally good behavior of the developed H-L-PN-based logic control structures.

From a monitoring point of view, the main characteristic of the proposed H-L-PN-based control structures is, that they allow clear visualization of the modeled flexible production systems evolution. The net can be considered as an important information source about the process developed in the system and also the behaviour of the components of the system and of the embedded discrete-event controller itself. Combining both, the information contained in the H-L-PN model of the logic controller and this coming from the process interface, a very good view of all logic and technical states and actions performed in the flexible production cell under control can be obtained. In this chapter, two sorts of monitoring methods have been presented and a structure for their implementation was proposed: feature- and model-based monitoring.

5 Real-Time Decision Level of a Hierarchical Control Architecture of Flexible Production Systems

To be able to show the concepts developed in this work, the attention is restricted to simplified flexible production systems, which are, for simplicity reasons, without certain complex cooperating behaviors as, for example, the case of two robots cooperating to carry the same load, or such that one of them is holding a part while the other one is performing machining operations on it. In addition, it is assumed that each resource can handle sequential activities only. As a consequence of these hypotheses, the H-L-PN-based logic control schema, developed in the last chapters, is restricted to include only sequential or alternative activities performed by each resource, and synchronization, mutual exclusion, and producer-consumer relationships among resources.

Nevertheless, the coordination and logic control components of the hierarchical DECS architecture proposed in chapter 2 are not sufficiently powerful to solve some problems generated in the field of control of flexible production systems. There are more situations in which a decision has to be made. Typical examples are processes with indeterminisms, i.e., material-flow and job release management – routing problems, path planning and collision avoidance, etc. The proposed H-L-PN-based control structure is unable to perform scheduling, and to handle anomalous behaviors, e.g., fault detection, disruptions on resource operations, material unavailability, error recovery, etc.

It is necessary to assist the developed control structures with another system with enough intelligence, both to help the coordination controller to update the state representation of the workshop in real-time and to make real-time decisions. First, operation research (OR) techniques have been widely used to support decision making in industrial production. However, difficulties have been encountered with the formulation of models, management of data, and interpretation of results. Since many problems of flexible production systems are usually unstructured or ill-structured problems that deal with non-numerical or non-algorithmic information, new methods have to be searched for /31/, /34/, /67/, /73/, /75/, /113/, /116/.

The proposal In the framework of this research work, is to incorporate a new component to the hierarchical DECS architecture, i.e., a real-time decision support system (RTDSS) for performing dispatching and diagnosis/error recovery functions together with the evolution of the H-L-PN-based control system.

5.1 Necessity of a Real-Time Decision Support System for H-L-PN Controlling of Flexible Production Systems

As depicted in Fig. 39, the core of the DECS is a feedback control loop between coordination and logic control component, monitoring and a RTDSS, which has to ensure the system stability, robustness, and performance. The proposed feedback turns the DECS into an investigation/decision structure /31/, /35/. All uncertainty and unexpected events which can not be solved by the lower components are left to the knowledge

incorporated in the RTDSS and to the feedback control loop to handle. Based on signal and information processing, the DECS allows one to easily formulate the control strategies, by highlighting the decision mechanisms and makes possible quick adaptation of control policies to real-time changes.

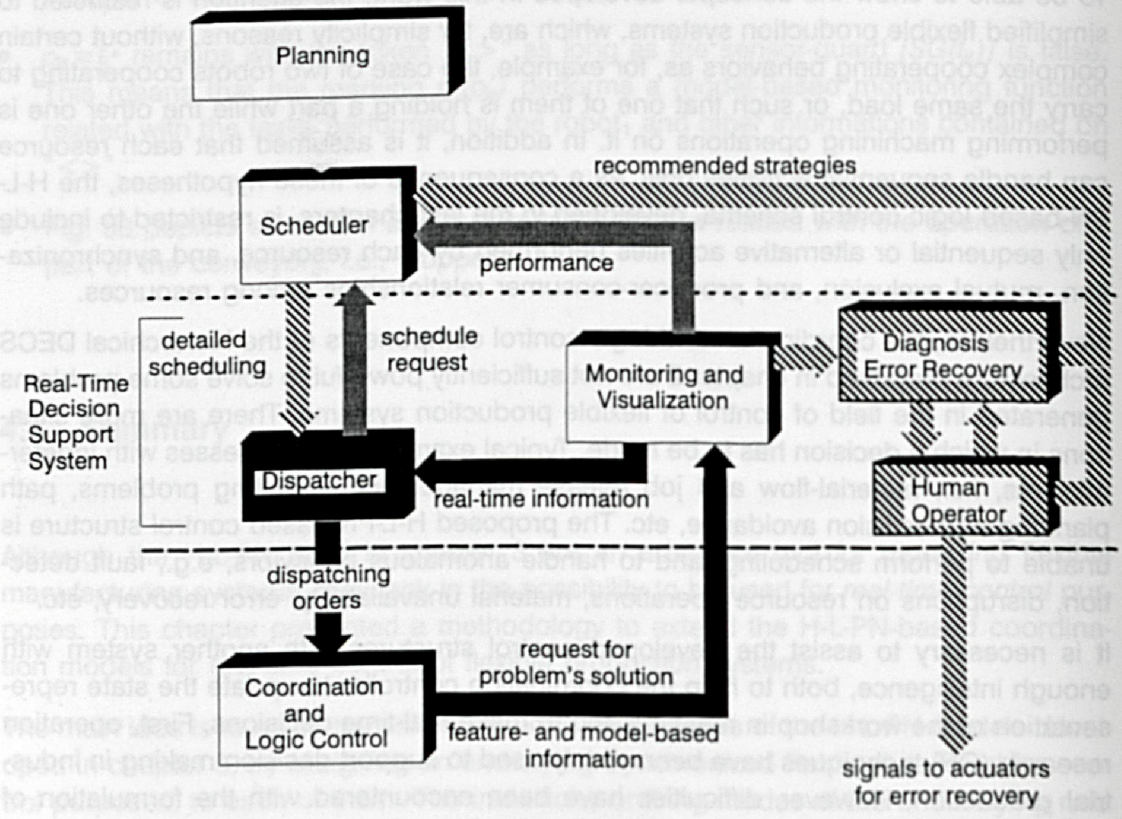


Fig. 39: Relation between Coordination and Real-Time Decision Support System

In order to accomplish:

- the allocation of FPC's resources in an efficient form,
- the movement of parts to resources in a way that reduces unnecessary idle time of both, parts and resources,
- the loading of parts into the system at scheduled times,
- the selection of the proper sequence of machines a part has to visit, and the selection of the times at which the parts visit the machines,
- the comparison of the set of the operations which are possible (because the required resources are free in the shop) with the set of operations which have to be done in order to respect the production schedule and to make the right decisions in real-time while also guaranteeing that production requirements are met,
- the detection of abnormal behaviors and possible quick adaptation of control policies after diagnosis functions are performed,
- the selection and setting into operation of error recovery strategies,

a proper distribution of functionality between both coordination and real-time decision support system, is important and this chapter has intention to explore this when the lower levels of the control architecture are described with a H-L-PN-based approach as shown in chapters 3 and 4.

As stated in chapter 3, operations such as the work of robots or the movement of pallets are modelled by transitions or their occurrence-modes, and the states of the resources in a flexible production cell are modelled by places and their colored markings in the High-Level Petri Net.

It is often the case that more than one operation can be performed in a production environment at the same time. In this case more than one transition, i.e., occurrence-modes, are enabled with respect to the marking of the net. Basically there are two possibilities: either the transitions are allowed to fire simultaneously, or they are not. This depends on the layout of the cell represented by the structure, and the marking of the H-L-PN model.

The states of a production cell related with the second possibility are identified as "conflict situations" during the evolution of the H-L-PN, and their existence is revealed by means of the structural analysis of the coordination control model /17/, /21/. The structure of the nets and the information associated to their components are sometimes not sufficient for solving the indeterminism modeled by such conflicts.

In this work two kinds of problems, not solved at the H-L-PN-based Coordination Control System, will be addressed and a methodology for solving them is proposed below: allocation of shared resources and material-flow specifications, and handling of errors and failures produced during the real-time operation of the flexible production system.

5.2 Allocation of shared Resources and Problems associated with Material-Flow Specifications

By means of examples, taken from the flexible production cell in Fig. 17, a case of concurrency and another typical situation of conflict will be shown below.

The first example depicted in Fig. 40 considers the work of two robots at the same time. The robot 1 is loading the work position of the human operator with a part type 2 and the robot 2 is processing a part type 1. Both operations are performed independently from each other. Therefore, transition t2 is enabled with regard to the occurrence-color $(m_2 \wedge p_2)$ and concurrently, transition t4 is enabled with regard to the occurrence-color $(m_1 \wedge p_1)$.

The second example consists on a decision-making process related with a set of possible operations, which can be performed by the robot 1. In this case, a decision must be made on which operation has to be done, i.e., the robot 1 can perform one, and only one, operation at a time. For example, a part type 1 is already processed by the robot 2 and its work position can be unloaded. At the same time, the human operator is free and its work position can be loaded with a part of type 2 or 3 that are on the corresponding input positions of the cell. As shown in Fig. 41, transition t1 is enabled

with regard to the occurrence-modes (<m_2,r_1,p_2,<•>>) and (<m_2,r_1,p_3,<•>>), and transition t5 is enabled with regard to the occurrence-mode (<m_1,r_1,p_1,<•>>).

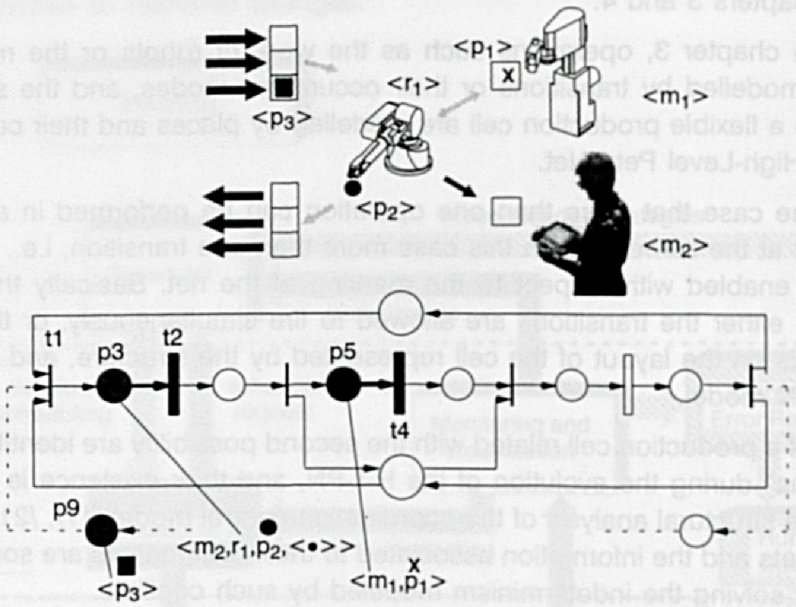


Fig. 40: Concurrency of Operations

This state of the cell presents the following typical indeterminism (conflict) situations:

- The occurrence-modes of transition t1 are in conflict with respect to the marking colors <r_1> and <m_2> (the robot 1 and the human operator are shared resources with respect to the operation "loading").
- Both occurrence-modes of transition t1 are also in a conflict situation with respect to the marking colors <p_2> and <p_3> (the parts of type 2 and 3 compete for being processed).
- the transitions t1 and t5 are in conflict with respect to the marking color <r_1> (the robot 1 is a shared resource with respect to the operations "loading work position of human operator" and "unloading of robot 2")

5.2.1 H-L-PN-Based Formal Specification of the Problems

From the analysis of the examples presented above, two kinds of conflicts can basically be distinguished: *structural*, ones related to the H-L-PN model, as in the last example, and *behavioral* ones, related to physical constraints of the flexible production cell layout and to the specifications of the operation plan to be developed in it.

Precise formulation of both kinds of conflicts is now given, in order to make this work self-contained.

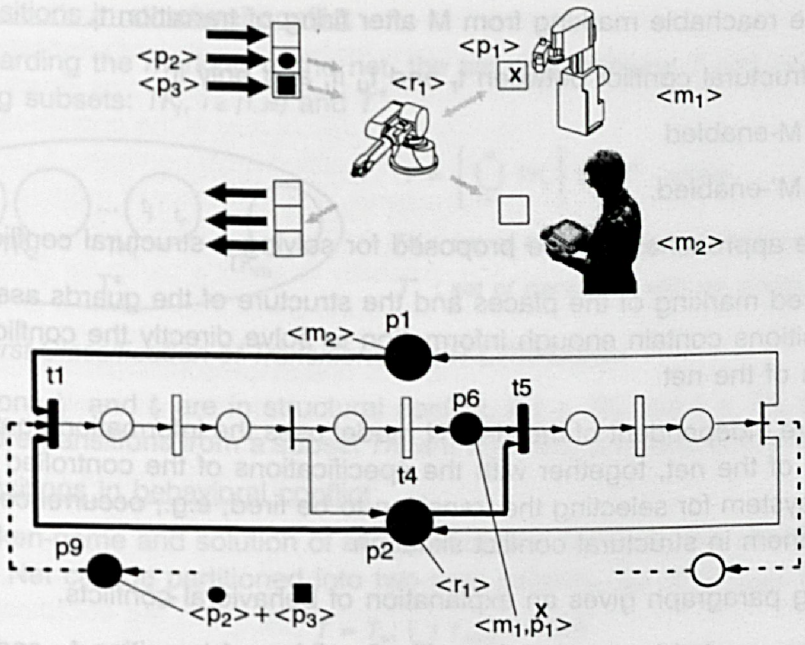


Fig. 41: Conflicts between Operations

The structure of the place p2, having two output transitions t1 and t5 concurrently enabled with regard to the same marking color, as shown in Fig. 42, is referred to as a *structural conflict*, *decision*, or *choice,* between marking-enabled transitions, depending on the application /74/.

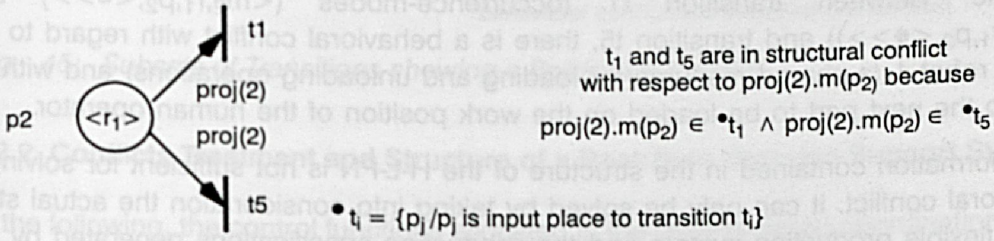


Fig. 42: Structural Conflict between Transitions

The structure of the place p1, having one output transitions t1 with two occurrence-modes concurrently enabled with regard to the same marking color, as shown in Fig. 43, is referred to as a *structural conflict*, *decision*, or *choice*, between marking-enabled occurrence-modes, depending on the application /59/.

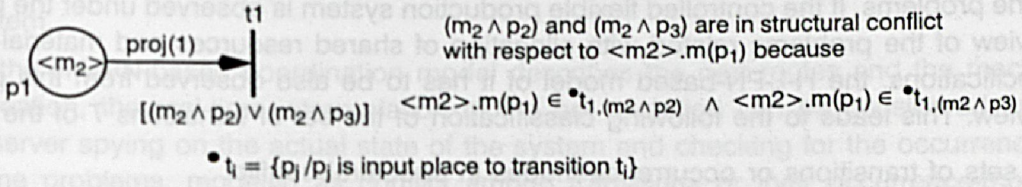


Fig. 43: Structural Conflict between Occurrence-Modes of a Transition

Let M' be the reachable marking from M after firing of transition t_r.

There is a structural conflict between t_r and t_u if, and only if:

- t_r, t_u are M-enabled
- t_u is not M'-enabled.

Two possible approaches can be proposed for solving a structural conflict:

- the colored marking of the places and the structure of the guards associated with the transitions contain enough information to solve directly the conflict during the evolution of the net
- a structure independent of the H-L-PN model uses the information contained in the structure of the net, together with the specifications of the controlled flexible production system for selecting the transition to be fired, e.g., occurrence-mode, from a set of them in structural conflict situation.

The following paragraph gives an explanation of behavioral conflicts.

Let M' be the reachable marking from M after firing of transition t_i, and M'' be the reachable marking from M after firing of transition t_r. Then, there is a behavioral conflict between t_i and t_r, if, and only if:

Both transitions model the work from two resources which compete for exclusive permission to use a shared resource /32/.

For better understanding of this kind of conflict, the case depicted in Fig. 41 serves an example. Between transition t1 (occurrence-modes $(<m_2,r_1,p_2,<\bullet>>)$ and $(<m_2,r_1,p_3,<\bullet>>)$) and transition t5, there is a behavioral conflict with regard to the use of robot 1 (a shared resource for loading and unloading operations) and with regard to the next part to be loaded on the work position of the human operator.

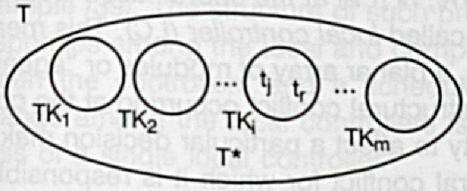
The information contained in the structure of the H-L-PN is not sufficient for solving a behavioral conflict. It can only be solved by taking into consideration the actual state of the flexible production system and the production specifications generated by the scheduling and planning components situated at the upper level of the hierarchical DECS, depicted in Fig. 39.

According to the definitions mentioned above, this new point of view of the H-L-PN-based models is now emphasized. The question is, indeed, to select the model entities relevant to the problems, detected in the coordination control level, e.g., transitions or occurrence-modes, and then to classify such entities in order to formalize the treatment of the problems. If the controlled flexible production system is observed under the point of view of the problems related with allocation of shared resources and material-flow specifications, the H-L-PN-based model of it has to be also observed from this point of view. This leads to the following classification of the set of transitions T of the net:

- sets of transitions or occurrence-modes in structural conflict
- sets of transitions or occurrence-modes in behavioral conflict.

Sets of transitions in structural conflict

Without regarding the marking of the net, the set of transitions T can be split up into the following subsets: TK_i, $i \in [1..m]$ and T^*.



$$T = \left\{ \bigcup_{i=1}^{m} TK_i \right\} \bigcup T^* \text{ , where}$$

TK_i : *set of transitions in structural conflict*

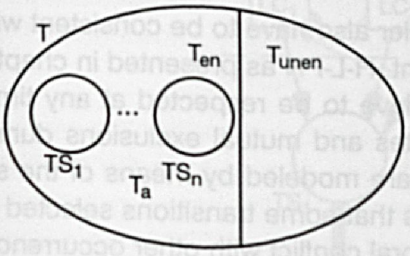
T^* : *set of transitions with no structural conflict*

Fig. 44: First Classification of Transitions in a H-L-PN Model

The transitions t_j and t_r are in structural conflict, if $t_j \in TK_i$ and $t_r \in TK_i$ (see Fig. 44). If two or more transitions from a subset TK_i are enabled, only one of them can be fired.

Sets of transitions in behavioral conflict

After the token-game and solution of all the structural conflicts, the set of transitions T of the Petri Net can be partitioned into two new subsets, T_{en} and T_{unen} (see Fig. 45).



$$T = T_{en} \bigcup T_{unen} \text{ , where}$$

T_{en} : *set of enabled transitions*

T_{unen} : *set of not enabled transitions*

$$T_{en} = \left\{ \bigcup_{i=1}^{n} TS_i \right\} \bigcup T_a \text{ ,where}$$

TS_i : *subset of enabled transitions, that model behavioral conflicts, e.g., physical restrictions*

T_a : *subset of all enabled transitions, that are not in behavioral conflict and therefore can be fired*

Fig. 45: Subsets of Transitions showing a Behavioral Conflict

5.2.2 Conflicts Treatment and Structure of a Real-Time Decision Support System

In the following, the control functions allocated in the H-L-PN-based coordination model are partitioned in order to distinguish the execution functions, i.e., communication between coordination and logic control as presented in chapter 4, from the decision-making functions, i.e., communication between coordination and real-time decision support system for solving conflicts.

The approach proposed here builds a new H-L-PN-based representation of the production shop, but, considering the same H-L-PN-based coordination model. The result is a problem-oriented interpretation of the H-L-PN-based model of the flexible production system.

As the H-L-PN-based coordination model describes the part routes and the machine allocation, the real-time interpretation of the net evolution can be considered as an observer spying on the actual state of the system and checking for the occurrence of some problems, modeled as conflict among transitions or their occurrence-modes, during the design phase of the coordination controller.

First step: all possible structural conflicts in the H-L-PN-based coordination model of a flexible production cell that are relevant for the job-release and job-flow control have to be captured. The result is a planar array of problems modeled as structural conflicts. Each structural conflict is represented by a set TK_i, $i \in [1..m]$ at the coordination level and mapped into a module allocated in the RTDSS, called *local controller (LC)*. This means that the RTDSS posses, among other structures, a planar array of modules or "agents", each of them with the intelligence for solving a structural conflict occurring at the coordination level. Each agent provides the capability to affect a particular decision making function related with the solution of the structural conflict for which it is responsible.

The real-time decision of each local controller has to be consistent with schedule, explicit or implicit schedule, which can be given either by time intervals associated with the tasks (earliest starting time – latest starting time), by an ordering (task A has to precede task B), by a set of rules (execute the shortest task first) or by a combination of the three /102/.

First result: one and only one occurrence-mode or transition of a set TK_i, $i \in [1..m]$ will be selected by each local controller to be enabled to fire in the coordination model.

Nevertheless, the real-time decision of a local controller also have to be consistent with the physical constraints of the production environment. H-L-PN as presented in chapter 3 and 4 describe the sequential constraints which have to be respected at any time, in any situation. These are, among others, part routes and mutual exclusions during resource utilization, and these specifications, which are modeled by means of the set of above defined behavioral conflicts. The last means that some transitions selected to be fired by a local controllers can be found in behavioral conflict with other occurrence-mode or transition selected to be fired by other local controller.

One possible way of solving the structural conflicts and the behavioral conflicts together is to provide the RTDSS with intelligence for handling complicated behaviors (conflict situations) via a federation of co-operating agents / local controllers. Each controller needs only to communicate with its immediate surroundings, but, at the same time, coordinates with neighboring controllers to produce larger-scale decision making control functions. Neighboring controllers can combine their data and information to obtain an overview beyond the scope of each of the group. According to Fig. 45 each federation is mapped into another type of *agent* in the RTDSS, called *complex controller (CC)*, which is responsible for coordinating the decision functions of the members in a hierarchical manner (see Fig. 46). Hence, the need arises of an overall perspective of a behavioral conflict, in which all related local controllers contribute to solve it. The solution of such a conflict will be found taking into consideration the knowledge about the actual state of the flexible production system, and the assistance of the upper level of the DECS, i.e., planning component with production objectives.

As result, for any given time instant, a complex controller situated in the RTDSS sends to the H-L-PN coordination controller information about the occurrence-modes and/or transition (one and only one per CC) that can be effectively fired according to its decision. This means, with this decision, an occurrence-mode or transition will be effectively

enabled to fire in the coordination model and the corresponding control enforcement will be issued from the last to the components of the production environment.

The conflict resolution in the RTDSS is performed by means of a protocol, see as example /39/. The structure of such protocol is based on an exchange of requests and responses among the local and complex controllers, involved in the topology, and between the controllers and a scheduler component. Additionally, an information exchange among the local controllers is necessary to support the decision-making process of a single local controller.

Based on the concepts mentioned above, conflict handling mechanism is performed in a hierarchical form. By this means, the structural conflicts are always solved first. Afterwards, the behavioral conflicts which emerge from the first step are treated.

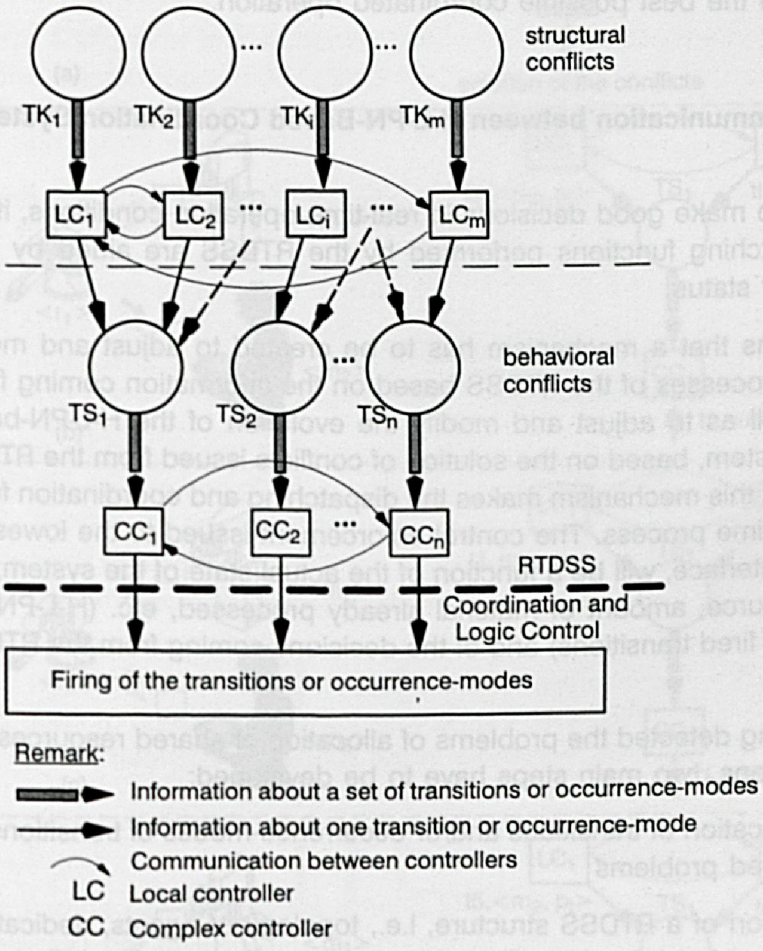


Fig. 46: Hierarchical Treatment of Conflicts

The topology of *co-operating agents* metaphor is regarded as being representative of the interactions and sharing of expertise that takes place when conflict situations are generated at the production environment level.

In keeping with good decision making processes each agent, i.e., local and complex controllers, would exhibit a high degree of cohesion. The community of agents would

be closely coupled, according to the topology, necessary for solving the different conflict cases. Each agent would therefore consists of a local knowledge base, a deductive capability and a communication mechanism that would enable it to interact with other controllers in the topology and with the H-L-PN coordination model. Consequently each knowledge base would be smaller and the model which operates upon this corpus of knowledge would not require the same degree of sophistication.

As shown above, the community of agents would collectively work toward the solution of problems reported from the coordination control system providing mutual assistance. In concept, if communicating with other local controllers, with relatively simple control laws developed for each local controller, complicated behaviors can be solved. Solutions of complex decision-making processes will emerge, which are useful in order to achieve the best possible coordinated operation.

5.2.3 Communication between H-L-PN-Based Coordination System and RTDSS

In order to make good decisions in real-time operation conditions, it is necessary that the dispatching functions performed by the RTDSS are aided by knowledge of the shop floor status.

This means that a mechanism has to be created to adjust and modify the decision making processes of the RTDSS based on the information coming from the coordinator, as well as to adjust and modify the evolution of the H-L-PN-based coordination control system, based on the solution of conflicts issued from the RTDSS. More important is that this mechanism makes the dispatching and coordination functions a closed-loop real-time process. The control enforcement issued to the lowest level, i.e., to the process interface, will be a function of the actual state of the system, e.g., condition of each resource, amount of material already processed, etc. (H-L-PN marking and sequence of fired transitions) and of the decisions coming from the RTDSS, as illustrated in Fig. 39.

After having detected the problems of allocation of shared resources and material-flow specifications, two main steps have to be developed:

- identification of transitions and/or occurrence-modes of transitions which model the identified problems
- definition of a RTDSS structure, i.e., topology of agents, dedicated to solve structural and behavioral conflicts reported from the coordination system

and two communication channels have to be defined and implemented between the components of the proposed closed-loop control architecture.

Below, a topology for solving the conflicts generated during the evolution of the sample cell of Fig. 17 is presented in Fig. 47a. Fig. 47b, c and d depict three possible conflict situations with the corresponding solution, according to the production-path specifications shown on the left side of the picture.

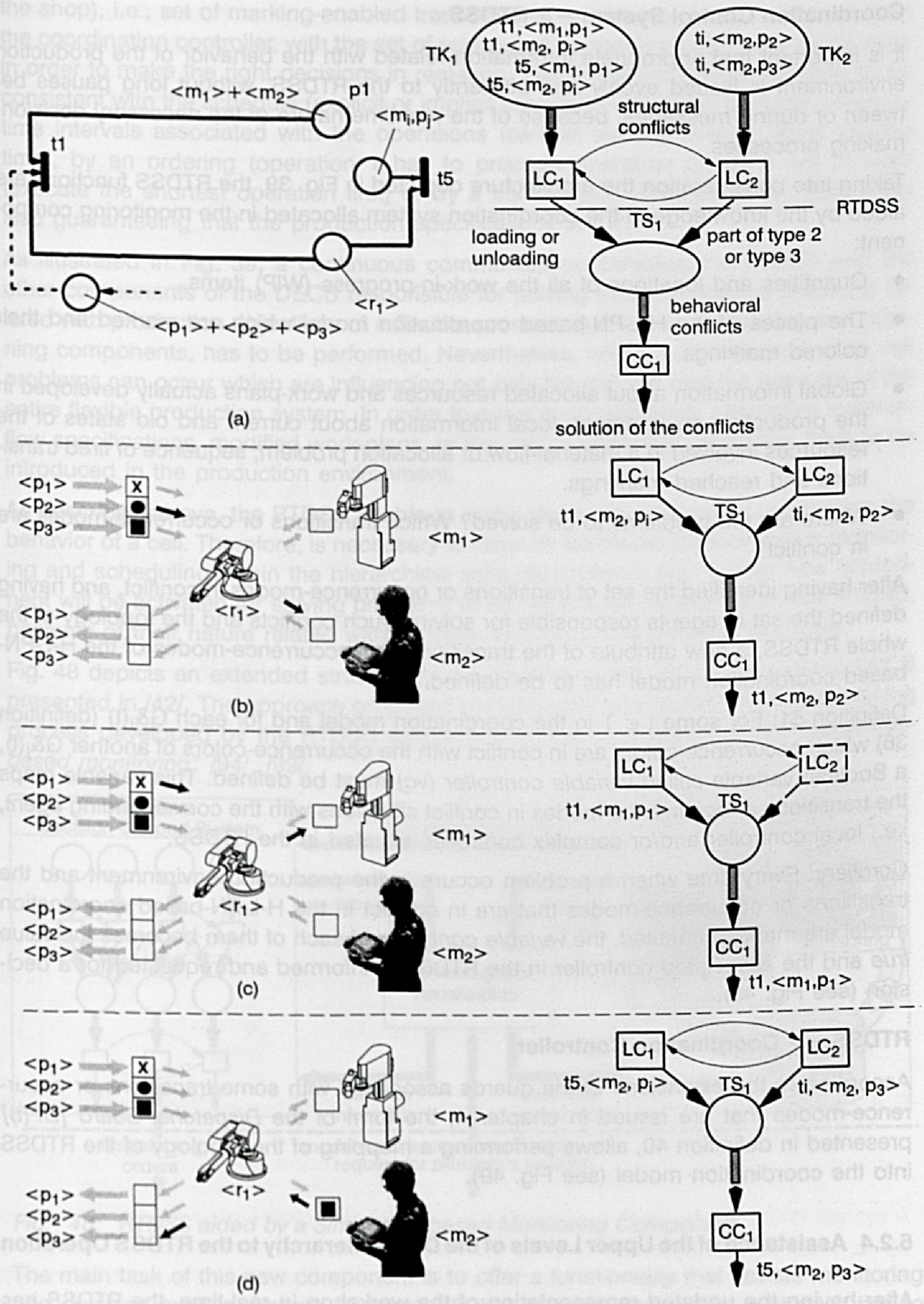


Fig. 47: RTDSS Topology for solving Conflicts in a Sample Flexible Production Cell

Coordination Control System –> RTDSS

It is important that appropriate information related with the behavior of the production environment is issued evenly and efficiently to the RTDSS, without long pauses between or during messages, because of the real-time nature of the necessary decision-making processes.

Taking into consideration the architecture depicted in Fig. 39, the RTDSS functions are aided by the knowledge of the coordination system allocated in the monitoring component:

- Quantities and locations of all the work-in-progress (WIP) items.
- The places of the H-L-PN-based coordination model which are marked and their colored markings.
- Global information about allocated resources and work-plans actually developed in the production environment; local information about current and old states of the resources involved in a material-flow or allocation problem; sequence of fired transitions and reached markings.
- Where are the problems to be solved? Which transitions or occurrence-modes are in conflict?

After having identified the set of transitions or occurrence-modes in conflict, and having defined the set of agents responsible for solving such conflicts and the topology of the whole RTDSS, a new attribute of the transitions and occurrence-modes of the H-L-PN-based coordination model has to be defined.

Definition 51: For some $t \in T$ in the coordination model and for each $G\&_i(t)$ (definition 36) which occurrence-colors are in conflict with the occurrence-colors of another $G\&_j(t)$, a Boolean variable called *variable controller (vc)* must be defined. This variable maps the transitions or occurrence-modes in conflict situations with the corresponding agent, i.e., local controller and/or complex controller, situated in the RTDSS.

Corollary: Every time when a problem occurs in the production environment and the transitions or occurrence-modes that are in conflict in the H-L-PN-based coordination model are marking-enabled, the *variable controller* of each of them becomes the value *true* and the associated controller in the RTDSS is informed and requested for a decision (see Fig. 49).

RTDSS –> Coordination Controller

According to the extensions of the guards associated with some transitions or occurrence-modes that are issued in chapter 4, the form of the *Dispatcher-Guard [DF(t)]* presented in definition 40, allows performing a mapping of the topology of the RTDSS into the coordination model (see Fig. 49).

5.2.4 Assistance of the Upper Levels of the DECS Hierarchy to the RTDSS Operation

After having the updated representation of the workshop in real-time, the RTDSS has to compare the set of possible operations (because the required resources are free in

the shop), i.e., set of marking-enabled transitions or occurrence-modes reported from the coordination controller, with the set of scheduled operations which have to be done in order to make the right decisions in real-time. These real-time decisions have to be consistent with the schedule (explicit or implicit schedule) which can be given either by time intervals associated with the operations (earliest starting time – latest starting time), by an ordering (operation 1 has to precede operation 2), by a set of rules (execute the shortest operation first) or by a combination of these three /102/, while also guaranteeing that the production specifications are met.

As illustrated in Fig. 39, a continuous communication between the RTDSS and the other components of the DECS responsible for solving decision making problems related with production objectives of a flexible production cell, i.e., scheduling and planning components, has to be performed. Nevertheless, within a flexible production cell problems can occur which are influencing not only the cell but also the behavior of the entire flexible production system. In order to solve such problems, alternative material-flow specifications, modified work-plans, or the use of additional capacities, have to be introduced in the production environment.

As described above, the RTDSS is able to make decisions exclusively concerning the behavior of a cell. Therefore, is necessary to have an additional component for monitoring and scheduling within the hierarchical schema shown in Fig. 1. This new component will be activated for solving problems which cannot be cleared up by the RTDSS, because of their nature related with medium and long time decision functions.

Fig. 48 depicts an extended structure of the decision levels of the hierarchical DECS presented in /42/. The approach proposed in this paper is based on a decision making process developed by the RTDSS assisted by a new component called *"simulation-based monitoring"* /94/, /99/.

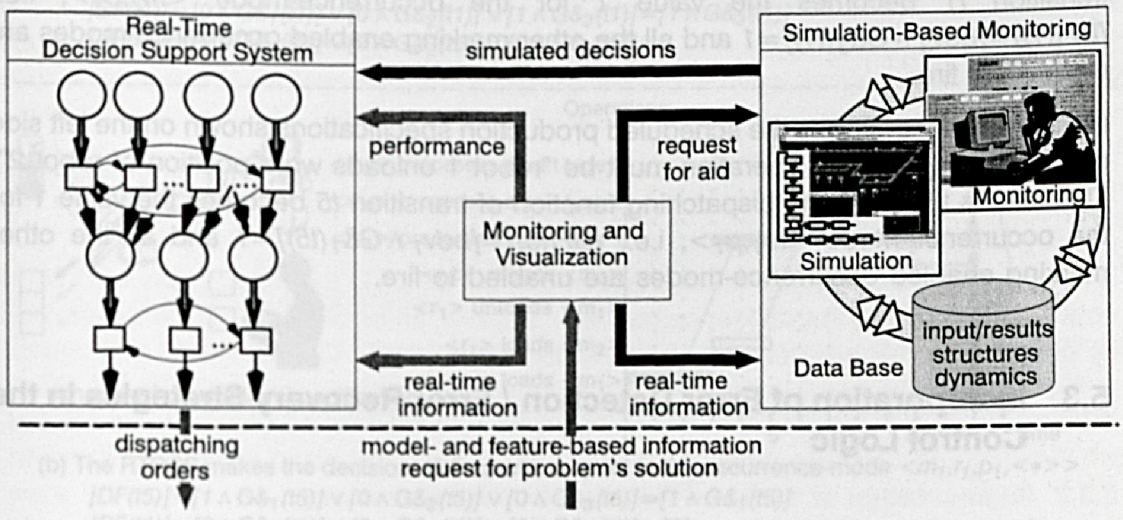


Fig. 48: RTDSS aided by a Simulation-based Monitoring Component

The main task of this new component is to offer a functionality that assists monitoring and decision making processes, parallel to the real time production system. It has to support the DECS in decision making processes not only with regard to a local opti-

mum on the level of the flexible production cells, but also with regard to the global optimum on the level of the production system. Therefore, the component has to: visualize user-oriented deviations and disturbances concerning planned processes, inform simultaneously about the consequences, and make suggestions to the RTDSS about possible reactions.

Basically, the processing and solution of a conflict can be summarized as follows: if the coordination controller detects problems, which have to be solved in the flexible production cell, they are reported to both, the RTDSS and the monitoring-based simulation component. The characteristics of the conflict, together with model-based and feature-based information related to the problem, are collected by the real-time monitoring structure (described in chapter 3) and sent to the RTDSS and to the simulation-based monitoring system. After processing the problems, i.e., conflicts, the solution is sent back from the RTDSS to the H-L-PN-based controller.

Fig. 49a and 49b illustrate two different results of a decision process that the RTDSS can make according to two production specifications generated by the scheduling and planning components of the hierarchical DECS, and to the actual state of the system reported by the H-L-PN-based coordination controller.

In both cases, the RTDSS, i.e., the complex controller CC_1, receives the information about the structural and behavioral conflicts detected between the transitions t1 and t5 and their corresponding marking-enabled occurrence-modes. The variable $vc=CC_1$ associated with the occurrence-modes of transitions *t1* and *t5* becomes the value *1*.

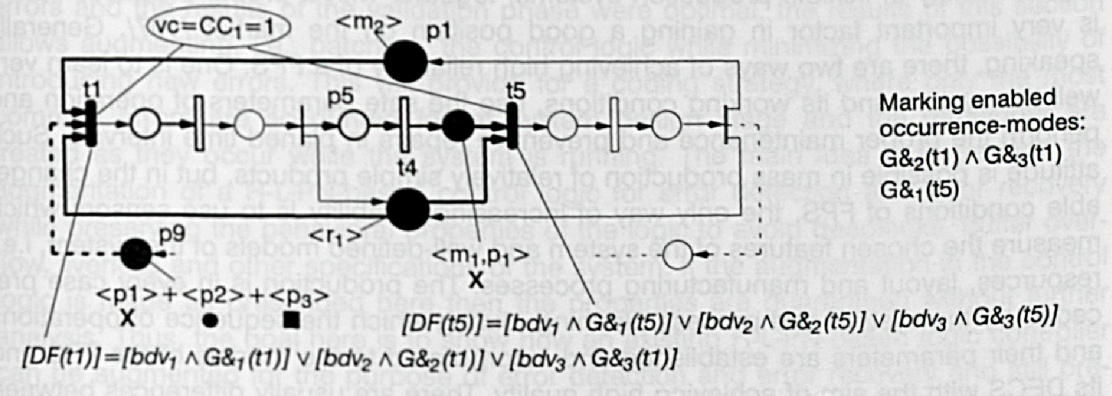
In Fig. 49a, according to the scheduled production specifications shown on the left side of the picture, the next operation must be "robot 1 loads the work position of the human operator with a part type 3". This means, that only the dispatching function of transition *t1* becomes the value 1 for the occurrence-mode $<m_2,p_3>$, i.e., $[DF(t1)]=[bdv_3 \wedge G\&_3(t1)]=1$ and all the other marking enabled occurrence-modes are unabled to fire.

In Fig. 49b, according to the scheduled production specifications shown on the left side of the picture, the next operation must be "robot 1 unloads work position of robot 2". This means that only the dispatching function of transition *t5* becomes the value 1 for the occurrence-mode $<m_1,p_1>$, i.e., $[DF(t5)]=[bdv_1 \wedge G\&_1(t5)]=1$ and all the other marking enabled occurrence-modes are unabled to fire.

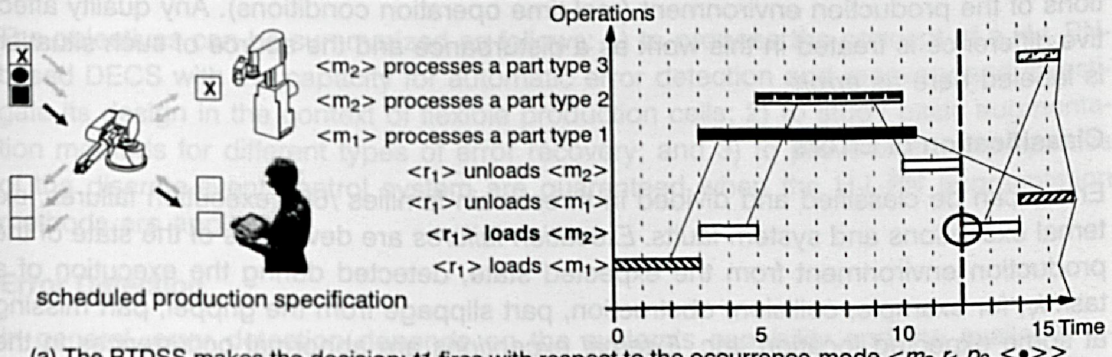
5.3 Incorporation of Error Detection / Error Recovery Strategies in the Control Logic

A system designed to perform a manufacturing process unattended, even when errors occur, must have a way of recovering from errors. Recovery comes after detecting the error and determining what it is (diagnosis). This section focuses on the detection step of the error recovery process in flexible production cells. It also formalizes an algorithm for constructing "error detection mechanisms", and proposes communication struc-

tures among control, monitoring and diagnosis components when the logic control structure of the DECS is based on a H-L-PN specification.



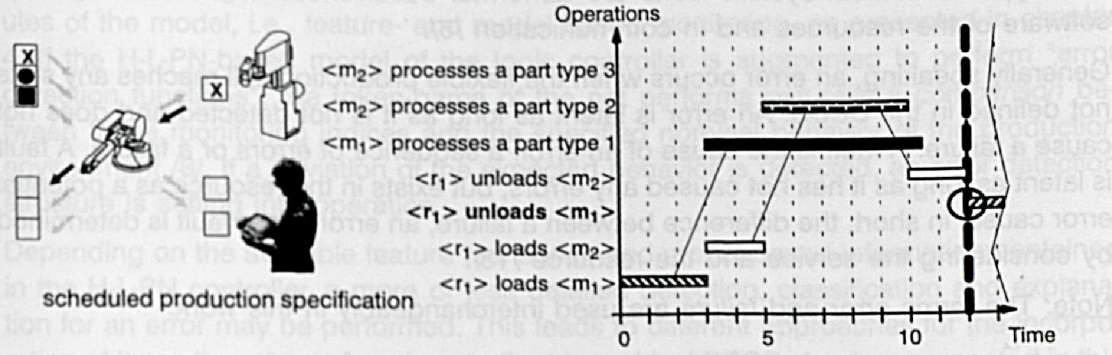
Marking of the H-L-PN-based coordination controller



(a) The RTDSS makes the decision: *t1* fires with respect to the occurrence-mode $<m_2,r_1,p_3,<\bullet>>$

$[DF(t1)]=[0 \wedge G\&_1(t1)] \vee [0 \wedge G\&_2(t1)] \vee [1 \wedge G\&_3(t1)]=[1 \wedge G\&_3(t1)]$

$[DF(t5)]=[0 \wedge G\&_1(t5)] \vee [0 \wedge G\&_2(t5)] \vee [0 \wedge G\&_3(t5)]=0$



(b) The RTDSS makes the decision: *t5* fires with respect to the occurrence-mode $<m_1,r_1,p_1,<\bullet>>$

$[DF(t5)]=[1 \wedge G\&_1(t5)] \vee [0 \wedge G\&_2(t5)] \vee [0 \wedge G\&_3(t5)]=[1 \wedge G\&_1(t5)]$

$[DF(t1)]=[0 \wedge G\&_1(t1)] \vee [0 \wedge G\&_2(t1)] \vee [0 \wedge G\&_3(t1)]=[0]$

Fig. 49: Decision-Making Processes for solving Structural and Behavioral Conflicts

5.3.1 Reliability of Flexible Production Systems Extension of Monitoring Functions of the DECS

High reliability of flexible production systems, together with a proper production plan, is very important factor in gaining a good position on the market /107/. Generally speaking, there are two ways of achieving high reliability of a FPS. One is to learn very well the system and its working conditions, use the safe parameters of operation and perform the proper maintenance and preventive repairs in planed time intervals. Such attitude is possible in mass production of relatively simple products, but in the changeable conditions of FPS, the only way of increasing reliability is to use sensors which measure the chosen features of the system and well-defined models of the system, i.e., resources, layout and manufacturing processes. The production is in every case preceded by the system and process planning, during which the sequence of operations and their parameters are established and incorporated to the models of the FPS and its DECS with the aim of achieving high quality. There are usually differences between parameters of the models considered during the planning phase and the actual conditions of the production environment (real-time operation conditions). Any quality affective difference is treated in this work as a disturbance and the source of such situation is labeled here as *error.*

Classification of Errors

Errors can be classified and divided in three main families /68/: execution failures, external exceptions and system faults. *Execution failures* are deviations of the state of the production environment from the expected state, detected during the execution of a task k_i, for example, collision, obstruction, part slippage from the gripper, part missing at some expected location, etc. *External exceptions* are abnormal occurrences in the flexible production cell which may cause execution failures. Misplaced parts, defective parts, and unexpected objects obstructing machine operations for instance, may cause such type of failures. *System faults* are abnormal occurrences in the hardware and software of the resources and in communication /8/.

Generally speaking, an error occurs when the flexible production cell reaches any state not defined in the DECS. An error is latent as long as it is not detected and does not cause a failure. A fault is the cause of an error, a sequence of errors or a failure. A fault is latent as long as it has not caused any errors, but exists in the resource as a potential error cause. In short, the difference between a failure, an error and a fault is determined by considering the service and the resource /13/.

Note: The terms *error* and *failure* are used interchangeably in this work.

5.3.2 Augmentation of the H-L-PN-Based Control Structures for Incorporation of Error Detection Functions

In implemented flexible production systems, an inordinate amount, up to 90%, of the control coding effort is dedicated to exception handling or automatic error recovery. At present, most of this coding effort occurs at the design stage. Engineers attempt to

anticipate common errors and write the control logic to handle them and to allow the system to automatically recover if feasible /117/.

By assuming that the H-L-PN-based control logic has been designed without coding errors and the results of the validation phase were optimal, the results of this section allows augmenting, i.e., patching, the control logic while minimizing the possibility of introducing new errors. This will provide for a coding strategy, where only the most common errors are accounted for, at system design stage and the remainder are treated as they occur while the system is running. The main idea is to perform the augmentation of a H-L-PN-based control logic for error detection and error recovery while preserving the behavioral properties of the logic to avoid deadlocks, buffer overflow, liveness and other specifications of the system. If the augmentation in the control logic is made as prescribed here then the properties are guaranteed without further analysis. Thus, the goal here is to show how an existing H-L-PN-based logic controller can be augmented for the purpose of error detection and error recovery and still preserve the desirable properties guaranteed in its initial design, as presented in chapters 3 and 4.

The objectives can be summarized as follows: 1) to propose the concept of a H-L-PN-based DECS with the capacity for automatic error detection and recovery and investigate its design in the context of flexible production cells; 2) to study basic augmentation methods for different types of error recovery; and 3) to prove that the properties of the discrete-event control system are guaranteed when the H-L-PN augmentation methods are applied.

Error Detection

In general, error detection depends on the system's capability and the available information. That is, error determination is specified as far as the system is capable. The monitoring functions performed naturally by the H-L-PN evolution is permanently acquiring monitoring indices from the raw sensor data and the static and dynamic attributes of the model, i.e., feature- and model-based monitoring, as presented in chapter 4. If the H-L-PN-based model of the logic controller is augmented to perform "error detection functions", the token-game of the net performs naturally a *comparison* between these monitoring indices and the specified nominal behavior of the production environment /8/. If a deviation of the expected behavior is detected, an error detection structure is setting into operation.

Depending on the available feature (sensorial) and model-based information contained in the H-L-PN controller, a more or less detailed detection, classification and explanation for an error may be performed. This leads to different approaches for the incorporation of these three basic functions to the hierarchical DECS structure presented in this work.

Basic Concepts of H-L-PN-Based Error Detection

Two detection modes are naturally implemented in the H-L-PN-based controller: *event monitoring* checks preconditions before the execution and goal achievement after the execution of the task. Alternatively, *continuos monitoring* can be performed checking

sensory conditions during the execution of a task /8/. According to these hypotheses, a task error can basically be detected through three kinds of symptoms:

- violation of the starting date for a scheduled operation;
- lack of signal from the process indicating the end of an operation;
- reception of a signal indicating a deviation of the process behavior.

Let $K=\{k_1,k_2,...,k_i,...,k_n\}$ be the set of tasks to be controlled by a H-L-PN-based DECS. Associated with each task k_i a time can be defined as $\tau_{Mi} \in \mathrm{R}^+$, which represents the maximum duration of it ($TM=\{\tau_{M1},\tau_{M2},...,\tau_{Mi},...,\tau_{Mn}\}$). There exists a function *temporization TEM:* $K \rightarrow TM$, whereby the value of each $\tau_{Mi} \in TM$ is fixed by production specifications.

The idea here is to incorporate to the original DECS structure an extended monitoring function that is used to detect non-nominal feedback in the system during the execution of each task k_i.

Taking into consideration the logic control architecture described in chapter 4, two main approaches to indicating of the occurrence of an error are proposed in the following. One is based on a combination of the information from the process interface of the flexible production cell, with the use of watchdog timers introduced in the structure of the H-L-PN controller /117/. The other one is based only on new information incorporated to the static structure and the evolution rule of the H-L-PN.

Augmentation of the H-L-PN-Based Logic Controller

As discussed in chapters 3 and 4, the approach to developing a H-L-PN-based logic controller is to use places of sub-nets to model technical actions, i.e., tasks k_i. The presence of a token in a place of a sub-net indicates the activity of a task, e.g., robot picks a part, lift moves a pallet, etc. The transitions of sub-nets are synchronized with external and instantaneous events, e.g., the start and completion of tasks execution.

In order to process abnormal states that appear during the execution of the system and detect errors using the H-L-PN control structure, a new function and a constant are attached to some transitions of the H-L-PN (sub-net) in the control logic structure. Also, a new basic color domain with the corresponding color function, new guards and new functionalities can be added to the models proposed in chapter 3 and 4.

In implementing a H-L-PN-based controller for a real-time system, time requirements also need to be satisfied. Therefore, it is necessary to introduce the time variable to the H-L-PN.

$\tau_i : T \rightarrow \mathrm{R}^+$

$\forall s \in \mathrm{N} \setminus \{0\} \Rightarrow \tau s_i : T \rightarrow \mathrm{R}^+ / \tau s_i = s.\tau_{Mi}$

$E=[e_1,e_2,...,e_i,...,e_n]$

These are explained as follows:

- The *elapse* and *safety-time*: $\tau_i(t_j)$ *and* $\tau s_i(t_j)$, $\forall t_j \in T$
  $\tau s_i(t_j)$ is the maximum amount of time, a marking-enabled transition remains without firing; $\tau s_i(t_j)$ is calculated taken into account the maximum duration of a modeled

task k_j to be controlled, i.e., τ_{M_j}, and a safety factor $s \in N \mid \{0\}$ to guarantee a reliable completion of it. $\tau_i(t_j) > \tau s_i(t_j)$ implies that an error may have occurred in the system (related with the task k_j) so that a *mechanism of detection* has to be activated.

- The standard color domain *"Error (E)"* which elements are all possible errors that can be modeled in the sub-nets with the corresponding mechanism of detection. The structure of the universal color domain, presented in chapter 2, has to be augmented with this basic color domain, and new standard functions will be defined.

Definition 52: The set E is now considered as a new Ω_j, i.e., a basic (standard) color domain, and its elements "color tones" are the possible errors that can occur in the production environment.

Definition 53: The universal color domain Ω^* of the net will be the cartesian product of all basic color domains defined during the modeling phase, as stated in chapter 3, and also the new $\Omega_j = E$.
That is $\Omega^* = \Pi_{j \in [1:n]} \ \Omega_j = \Omega_1 \times \Omega_2 \times ... \times \Omega_{(n-1)} \times \Omega_n$ then
$\forall\ \omega^* \in \Omega^* \Rightarrow \omega^* = <\omega_1,\omega_2,...,\omega_{(n-1)},\omega_n> = <\omega_1,\omega_2,...,e_i,\omega_n>$
Without loss of generality, in this work it is considered that $\Omega_{(n-1)} = E$.

Definition 54: Under normal operation conditions, the marking of the H-L-PN-based control system (sub-nets) contains as penultimate element the color tone e_0. It means that no error occurred during the evolution of the system until the current state.

Definition 55: Associated with the new extension of the universal color domain, a new set of functions can be defined, which will be used for constructing error detection structures in the sub-nets of the H-L-PN-based logic controllers. Examples of these functions are:

1) The projection function

$proj(n-1)$: $\Omega^* \rightarrow E$

2) The successor function

$succ((n-1)_x)$:$\Omega^* \rightarrow \Omega^*$: $\quad <\omega=(\omega_1,...,\omega_n) \rightarrow \quad (\omega_1,...,\omega_{(n-1)} \oplus x,\omega_n)>$

3) The predecessor function

$pred((n-1)_x)$:$\Omega^* \rightarrow \Omega^*$: $\quad <\omega=(\omega_1,...,\omega_n) \rightarrow \quad (\omega_1,...,\omega_{(n-1)} \ominus x,...,\omega_n)>$

4) It is also possible to build a composition of the above defined color functions. The following example shows a composition that allows to extract the characteristic of a modeled error from the whole structure of a token.
a) If $succ((n-1)_x)$:$\Omega^* \rightarrow \Omega^*$ and $proj(n-1)$: $\Omega^* \rightarrow E$ then
$proj(n-1).succ((n-1)_x)$:$\Omega^* \rightarrow E \Rightarrow proj(n-1).succ((n-1)_x)$:$\Omega^*$:$<\omega=(\omega_1,...,\omega_n) \rightarrow (e_i \oplus x)>$

Fig. 50 depicts a general augmentation of the H-L-PN-based logic controller to detect the presence of an error in the controlled flexible production system using a method to detect the absence of sensor signals.

If a place *pa* in a sub-net models the task k_j, a first kind of error detection structure related with this task is illustrated in Fig. 50. Timed transition is associated to the model so that the place *pa* is also pre-condition of the last. A new pair of places connected

with the timed transitions represent together with the transition the error detection mechanism for the considered task.

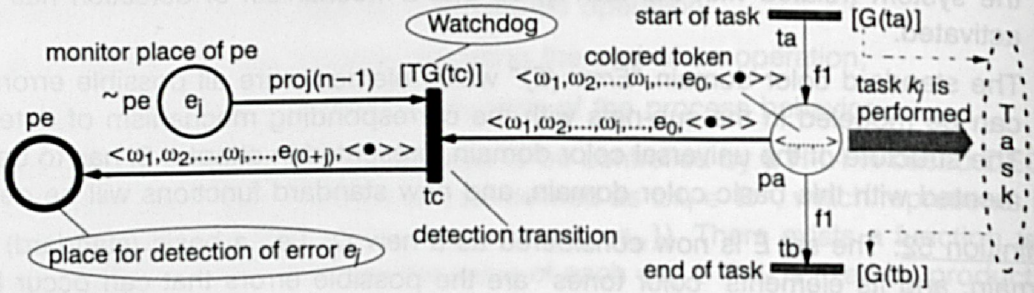


Fig. 50: Attributes of a H-L-PN Controller augmented for Error Detection

The greatest implication of this structure is that the properties of the sub-net are maintained, because both new places constitute a new place-flow of the sub-net and the transition generates a new transition-flow which models a new control logic sequence related with the detection of the error. In the schema of Fig. 50, following remarks can be done:

- $proj(n-1).m(pe)+succ((n-1)_j).m(\sim pe)=proj(n-1).m_0(pe)+succ((n-1)_j).m_0(\sim pe) =<e_j>$
  The marking of the places *pe* and $\sim pe$ are in mutual exclusion relation with regard to the token $<e_j>$. This means that the place $\sim pe$ remains marked during normal operation conditions of the production environment. As soon as the error e_j is detected, this places will be unmarked and place p_e receives the marking $<\omega_1,\omega_2,...,e_j,<\bullet>>$ which contains all the information that is necessary for performing the first steps of a diagnosis function.
- *[ta tb 0]* is a canonical transition-flow corresponding to the logic control sequence for normal operation conditions related to the task k_j
- *[ta 0 tc]* is a canonical transition-flow corresponding to the logic control sequence for operation conditions with error e_j related to the task k_j.

Definition 56: Associated with the standard guard of the new transition, a timer, i.e., watchdog, is defined. The value of this timer is fixed on τs_j. For the case depicted in Fig. 50, as soon as the place *pa* receives a token, the transition *tc* becomes marking-enabled condition, but it can not be fired because of its *Timer-Guard [TG(tc)]*.

At the same time, two synchronized events have to be started: the task k_j is performed in the real world and the watchdog of the transition begins with a countdown from the value τs_j to *0*.

Definition 57: Under normal operation conditions, i.e., without errors at the production level, the colored token resides in a place until the action it represents is completed, that is during the time τM_j. When the last event occurs, i.e., the completion of the task, the transition of the normal logic sequence, which pre-condition is the marked place, fires (atomic firing). The last event causes that the transition of the error detection structure is marking disabled.

Since the firing of a transition in a H-L-PN-based logic controller is synchronized with the production environment by means of sensor signals, the absence of at least one of such signals interrupts the normal token-game of the net. This leads also to the synchronized interruption of the next processes that could be started according to the control logic sequence and the initialization of a error detection and error recovery logic.

Note: The role of the watchdog is to make sure that the place of the original logic sequence does not remain marked for a duration exceeding that specified for the modeled task.

After a time τs_j, if the tasks k_j can not be completed because of an error, failure, breakdown, etc. in the production environment, the production process related with this task has to be stopped and the control logic updated. The transition with the watchdog becomes effectively enabled and fires. This event initializes a new control logic sequence, i.e., the normal control logic is desactived and a error detection sequence is concurrently started.

Fig. 51 shows an alternative structure for performing error detection. In this case, a simplified date structure is considered for the tokens to be used in the detection part of the controller.

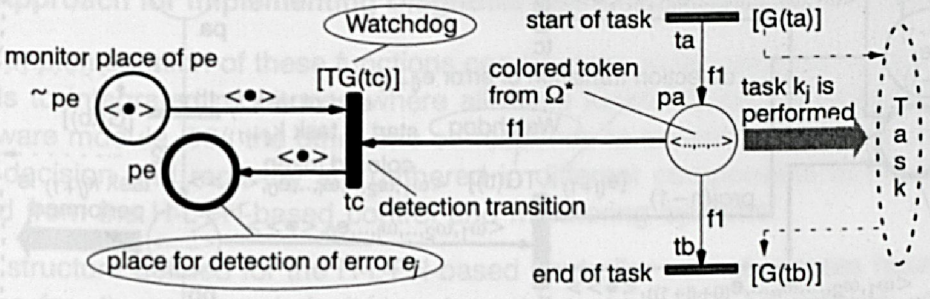


Fig. 51: Structure of a H-L-PN Controller augmented for Error Detection

Another possible mechanism for detecting errors in the production environment combines a non-timed H-L-PN-based detection structure with the identification of features performed in the production environment and reported in real-time by means of sensorial information.

Definition 58: If the modeled detection mechanism reports an error e_j corresponding to the task k_j, the marking of the H-L-PN model has to be locally actualized and the penultimate position of the involved colored token becomes the color tone e_j (see Fig. 52). That is $succ((n-1)_j) < \omega = (\omega_1,...,e_0,\omega_n) \rightarrow (\omega_1,...,e_0 \oplus j,\omega_n) >$.

Since a sub-net represents the control logic related with a set of tasks that are necessary for performing an operation in the flexible production system (see chapter 4), for each task k_j a similar error detection structure as the presented above can be defined.

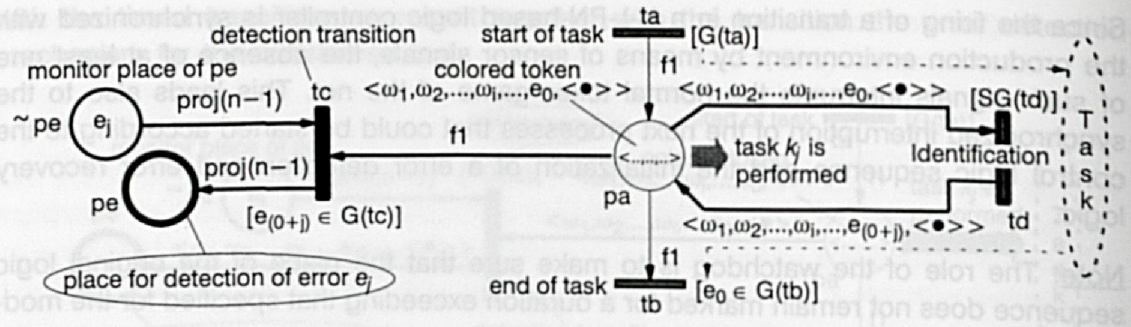


Fig. 52: Error Detection combining Sensor Signals and Attributes of a H-L-PN

Below, the proposal is to incorporate a pair of places to each sub-net and a detection transition for each task to be monitored. Fig. 53 depicts an example of two tasks to be performed sequentially and monitored with timed transitions as presented in definitions 56 and 57.

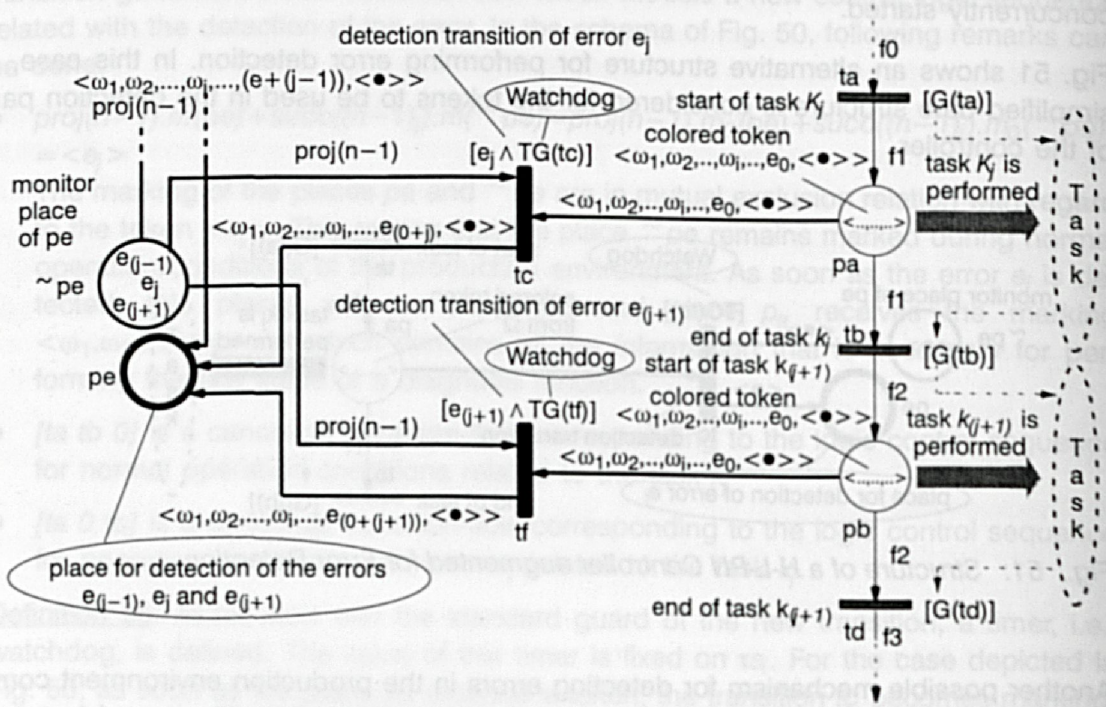


Fig. 53: Error Detection in a Sub-net of a H-L-PN Controller

5.3.3 Augmentation of the DECS Structure for Incorporation of Diagnosis and Error Recovery as Supervisory Functions

The detection mechanism proposed in the last section can be summarized as follows: when the flexible production system works normally, the sequences represented in the original H-L-PN-based control logic will be executed. Otherwise, as an error occurs, the detection structure associated to each task to be monitored will be started. But monitoring (detection) is only the first step of a supervisory activity for error recovery /107/.

After detection of an error, the DECS system should perform a diagnosis function and then influence the operation of the flexible production system eliminating or at least diminishing losses which may occur. The last means that four steps can be distinguished during the error processing: detection, diagnosis, decision (the selection of an optimal error recovery strategy) and recovery.

- The diagnosis function will firstly check if there really is an error (error confirmation) and update the internal model of the DECS. Then, this function will try to classify and explain the error. At each execution level, different levels of explanation for a detected error may be generated, depending on the amount of available information /7/, for example, a gross diagnostic can be "transport_1 fail". A more detailed diagnostic could be "transport_1 fail due to motor failure".
- According to the classification made in /117/ and /43/, depending on where the correct control logic sequence will restart after an error has been detected, classified and an error recovery strategy has ben selected, it is possible to distinguish three possible automatic error recovery strategies:

+ Backward recovery
+ Forward recovery, and
+ Redo recovery

An Approach for Implementing Diagnosis and Error Recovery Functions

The implementation of these functions can be done in two different forms: the first one leads to integrated monitoring where all these functions are implemented in a single software module /96/, the other one corresponds to separated modules where diagnosis, decision and recovery are gathered in different components of the DECS, separated from the H-L-PN-based control and monitoring systems.

The structure defined for the H-L-PN-based controller cannot express heuristic mechanisms for diagnosis and decisions. In order to keep the advantages of the normal control sequences of a flexible production system by means of H-L-PN, and to deal with diagnosis and decisions, it is necessary to combine the information of the processes contained in the nets with artificial intelligence techniques /13/, /71/, /75/. An architecture for an intelligent automatic H-L-PN-based error detection and and automatic and/or manual error recovery system that work together with the first one in a synchronized form is shown in Fig. 39.

Remark: It must be pointed out that for the diagnosis rules the marking of the nets are important facts. Consequently, a great advantage of the proposed H-L-PN-based DECS is to allow an easy and clear access to "states" (control system states or manufacturing system states) and "actions" (operations and tasks in the production environment or control enforcement located at the DECS level).

In order to process the information contained in the H-L-PN-based error detection structure and to incorporate diagnosis and error recovery functions to the DECS, a similar construction as the proposed for the RTDSS is now presented.

As a matter of fact, a diagnosis component has to be built, which is composed of a set of agents, i.e., error controllers, each of which is responsible for the treatment of one of the detected errors that have been modeled in the H-L-PN logic controller.

Let $EC=\{ec_1,ec_2,...,ec_i,...,ec_n\}$ be the set of agents, i.e., controllers, dedicated to process the errors of the set E defined in section 5.3.2.

Fig. 54 depicts a possible extension of the H-L-PN-based error detection structure for performing "error classification" and "communication" with the upper level of the DECS. Two new transition are incorporated for building the communication between the H-L-PN-based control logic component and the diagnosis component of the DECS.

The transition t_{diag} models the communication between H-L-PN-based logic controller and diagnosis component and the transition t_{rec} models the communication in the opposite way.

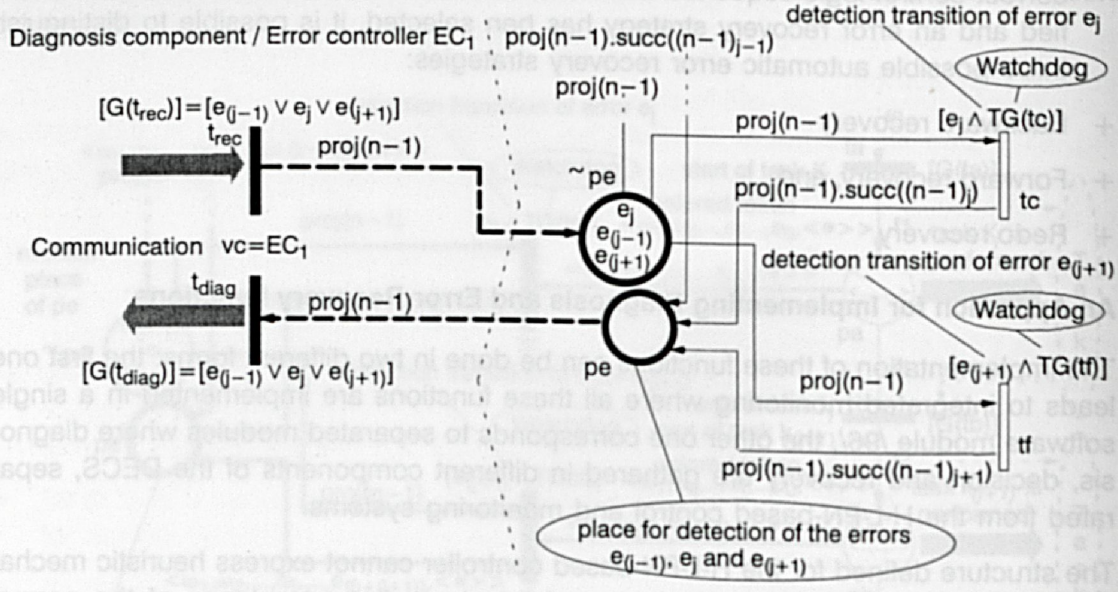


Fig. 54: Classification of Errors and Communication with a Diagnosis Component

The transitions t_{diag} and t_{rec} possess as many occurrence-modes as errors can be detected by the place *pe*. The agent or agents responsible for performing the diagnosis functions will be defined as attributes of a variable controller associated with these transitions, similar as explained in the case of the RTDSS.

Remark: The exchange of signals and information between the H-L-PN-based logic control system and the diagnosis component is performed by using the same "conflict oriented" approach proposed in section 5.2.3 for solving problems related, for example, with the allocation of shared resources.

As proposed in Fig. 39, the diagnosis and error recovery component of the DECS receives the information about the detected errors plus all the other information contained in the monitoring component. The array of agents, i.e., error controllers, situated in it, processes the information and selects and implements a recovery strategy (solution of the conflict). It is possible for the application of a full automatic recovery to use

some of the techniques above referenced and influence the decisions made by the RTDSS, or the intervention of the human operator for directly actuating at the real production system (Note: A detailed discussion about the last processes is beyond the scope of this work).

5.4 Summary

The analysis of the H-L-PN-based coordination control structures developed in chapter 3 revealed the existence of conflicts, which refer, among others, to problems generated during the flexible production system evolution, e.g., the use of shared resources and competence relationships between components. Furthermore, the proposed control structures do not allow these conflicts to be solved at this control level.

Taking into consideration the hierarchical discrete-event control architecture proposed in chapter 2, this chapter aims at the design of a real-time decision support system (RTDSS) and the interface between it and the underlying H-L-PN-based coordination and logic control systems.

The approach comprises the modularization of the coordination model that corresponds to the layout of the modelled flexible production system. A basic RTDSS structure composed of a set of agents was proposed, after dividing the real production system into modules. It takes into account a hierarchical treatment and solution of conflicts at the coordination level.

New augmentation of of the control structures were presented in this chapter to perform error detection and to allow the incorporation of error recovery strategies to the developed controllers. This was possible after assuming that the given H-L-PN-based hierarchical DECS was designed while preserving the structural and behavioral specifications of the FPS.

The complexity of the proposed real-time decision level is justified in terms of user requirements, as it allows maximal system operation flexibility according to the specified work-plans. Additionally, the new introduced error detection structures and the augmentation of the H-L-PN-based control systems, with error recovery functionalities, guarantee a high DECS reliability and the integration of the proposed control components into the proposed control hierarchy.

6 Implementation of H-L-PN-Based Control Structures of Flexible Production Systems into a PC-Platform

There are many requirements, which have to be taked into account for implementing flexible production systems FPS /5/, /91/, /108/, /110/, /117/. The most important are:

- unified management and control from overall production scheduling to on-line control of the smallest production unit;
- a high degree of expandability and maintainability in system hardware and software;
- a high level of system reliability and fault tolerance;
- minimum engineering man-power and short-term man-power for software system development and enhancement; and
- minimum system costs realized through the adoption of suitable implementation tools.

The H-L-PN-based formal synthesis of a discrete-event control system of FPS/FPC, such as described in chapter 3 and 4, allows meeting the above addressed requirements. It offers many important advantages over other currently reported design approaches:

- The structure of the DECS and the control logic embedded in it are correct by construction, because they are generated in accordance with structural and behavioral specifications of the controlled FPS.
- The functionalities of the hierarchical obtained control structure are maximally permissive within the considered specifications. These functionalities are, per construction, the same as the one of the real FPS/FPC.
- Control logics are easily comprehended and can be modified independently of other logics, facilitating control software enhancement and upgrades.
- The mapping between control sequencing information and the control code is straightforward and so the sequencing information is easy to see by just looking at the control code described in the H-L-PN structure. It also leaves the control sequencing information in an understandable and therefore more easily maintained and modified form.
- As processes or FPS/FPC requirements change, H-L-PN-based descriptions can be easily updated in an interactive mode or in the higher ranked system. Furthermore, the H-L-PN-based controllers are able to handle the addition of equipment and are relative easy transportable to other workstations.

Since the functionalities of the H-L-PN-based DECS and of the real production environment are the same, the H-L-PN-based DECS structure is defined now as a virtual FPS/FPC. That is, a virtual production environment composed of *software/mechatronic components* embedded in the structure of the net.

An entity *software/mechatronic component* is an extension of a software component concept /76/, /117/. Such a component has to include specifications of:

- components of the real production environment, e.g., machines, robots, transport systems, which are considered from the point of view of their mechanical specifications (number of transport places, different states of a robot, possible movement directions of a lift, layout specifications and port structures);
- components of the real production environment, e.g., machines, robots, transport systems, which are considered from the point of view of their electrical/electronic specifications (number of velocity-phases of a motor, number and type of sensor signals that can be handled by an identification component).

The point is, the production engineer has to consider the existence of two production worlds: *a virtual production environment,* constituted by the H-L-PN-based DECS and the *real production environment*, which evolves in a synchronized manner with the first one. As main result, both production worlds are provided as one loosely coupled system, are gradually integrated and tightened into a total flexible production system, i.e., the virtual and the real production environments are setting into operation as a unique production entity.

As described in chapter 2, the development of a hierarchical H-L-PN-based DECS for FPS is divided into three main steps:

1) Modeling and validation of the real production environment (structural and behavioral specifications to be enforced), development of the software/mechatronic components;

2) Synthesis of the virtual production environment, which has to feed to a computer program, implementation of the software/mechatronic components;

3) The implementation of the virtual production environment. It has to be coded in a computer-based platform (PC-based implementation) or in a Programmable Logic Controller-based platform (PLC-based implementation).

In order to perform the last step and considering both possible implementation platforms addressed above, this chapter proposes 3 approaches which have been implemented in a PC-based platform and set into operation at the FAPS laboratory. Here will be described computer aided design and manufacturing tools (CAD–CAM) to support a successful application of H-L-PN to industrial automation, and reasons for their wide acceptance by application engineers.

The set of developed and implemented user-friendly CAD tools ease the usage of the sophisticated H-L-PN theory for many industrial applications in manufacturing systems design and implementation, and help the generating of the virtual production environment defined above.

The H-L-PN-based engineering tool includes the following important components:

- powerful and user-friendly H-L-PN graphic editor;
- behavior analyzer via reachability graph generation, invariant method and performance evaluation via timed simulation;

- H-L-PN controller;
- 2-D-based monitoring/visualization system;
- interfaces necessary for binding the H-L-PN-based controller with the other components of a hierarchical DECS, and with a 3-D kinematic simulation tool for test functions.

The major tasks in the development and implementation of each component are outlined as follows.

6.1 Graphic Editor of H-L-PN for Control Purposes

A well-designed graphical user interface (GUI) is the key to successful application of the H-L-PN-based engineering tool. The development of a graphic editor gives a base for the construction of the control framework of the FPS. The result is universality of the system framework that offers a possibility of connecting the planing, realization and commission fields without changing of model, in this case the usual programming and control tasks are not necessary anymore.

Based on the theoretical concepts presented and proved in chapters 3, 4 and 5, two CAD-packages have been developed for the edition of H-L-PN-based control structures (see Fig. 55).

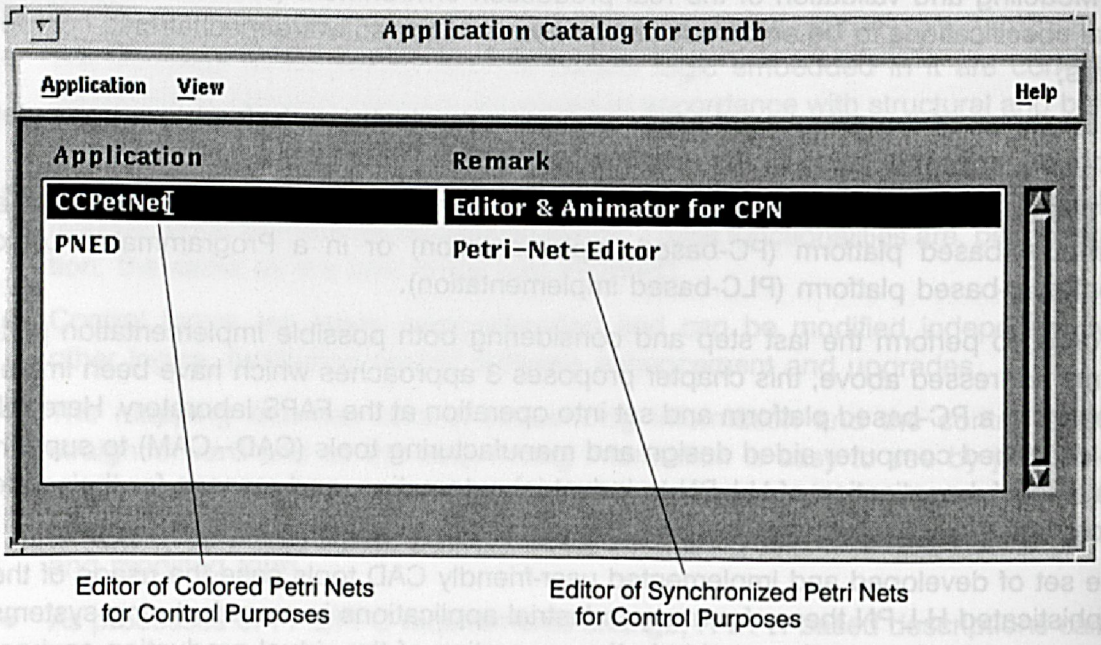


Fig. 55: Implemented Editors of Petri Nets for Control Purposes

Fig. 56 depicts a view of *"Petri-Net-Editor (PNED)"*, a first version of a CAD-tool that was developed for the edition of synchronized Petri nets. With this editor it is possible to generate Petri-Net-based logic control structures of simple FPS/FPC. The tokens of the edited nets are those of standard Petri Nets (i.e., uncolored tokens without any information about modeled components of the FPS).

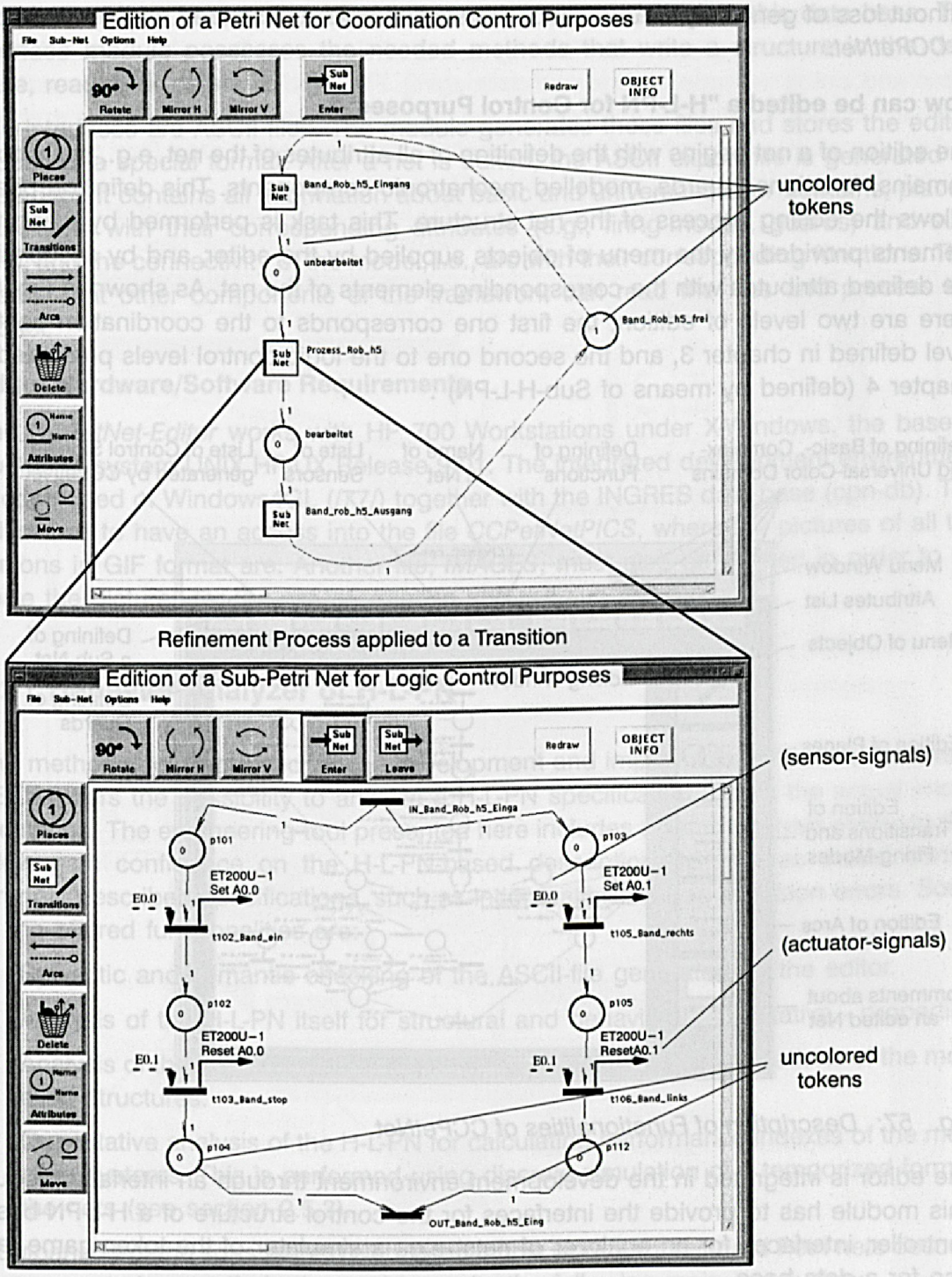


Fig. 56: View of PNED–Editor of Synchronized Petri Nets

The edition of logic control structures of complex FPS and the small set of functionalities offered by *PNED* is necessary, and therefore this first editor was enhanced to allow the edition of H-L-PN, e.g., Colored Petri Nets. The new version is *"Control-Colored-Petri-Nets (CCPetNet)"*, which capabilities complete those of the first version (see Fig. 57).

Without loss of generality, in the rest of this section will be described the functionalities of *CCPetNet.*

How can be edited a "H-L-PN for Control Purposes"?

The edition of a net begins with the definition of all attributes of the net, e.g., basic color domains, functions, guards, modelled mechatronic components. This definition phase follows the editing process of the net structure. This task is performed by using the elements provided by the menu of objects supplied by the editor, and by associating the defined attributes with the corresponding elements of the net. As shown in Fig. 57, there are two levels of edition, the first one corresponds to the coordination control level defined in chapter 3, and the second one to the logic control levels presented in chapter 4 (defined by means of Sub-H-L-PN) .

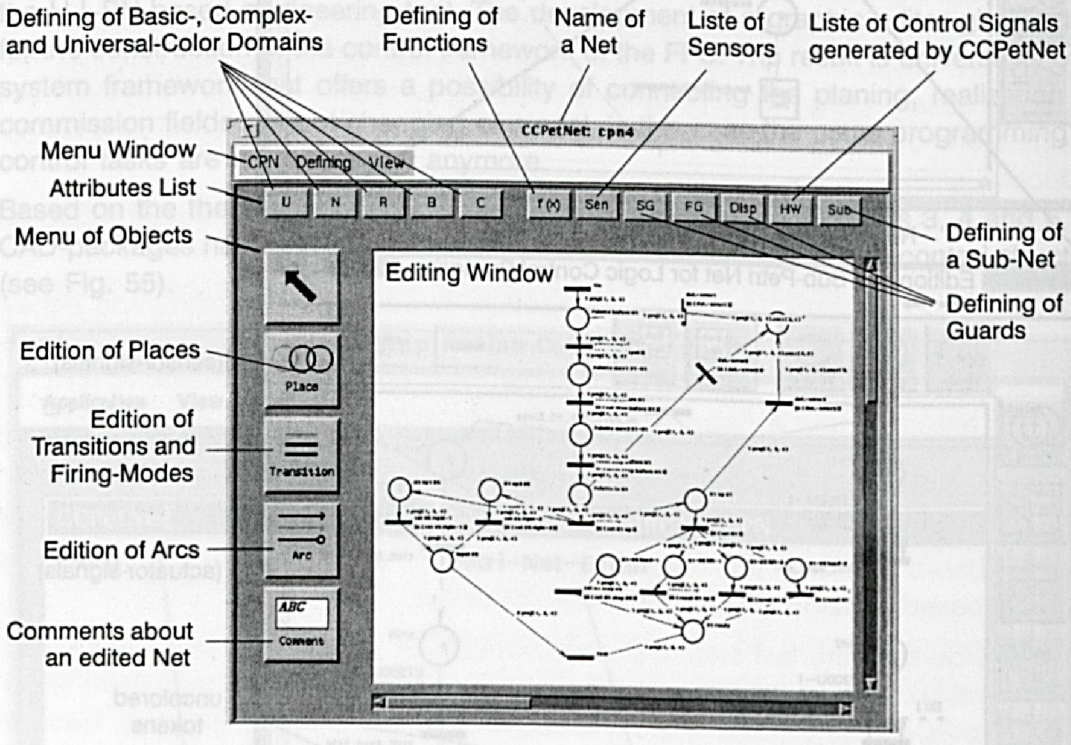


Fig. 57: Description of Functionalities of CCPetNet

The editor is integrated in the development environment through an interface module. This module has to provide the interfaces for the control structure of a H-L-PN-based controller, interfaces for an analyzer of nets, e.g., a simulator of the token-game, and also for a data base.

The models produced with the *CCPetNet* are to be converted in a data base with the help of a data base management system *(DBMS)*. This practically means, the models are stored in the data base and from this place they can be loaded and/or also deleted.

The interface between the *CCPetNet* and the *DBMS* must develop a corresponding data base language. The designer has to be able to store the edited model and load it again. For this purpose, a data base is used as a component of the development

environment /77/. The complete net description can be stored in this data base. The interface module possesses the needed methods that write a structure in the data base, read it, or delete it.

The interfaces are ASCII files. The module generates these files and stores the edited model as a special format. After a net is edited, the ASCII export file is generated by the editor. It contains all information about basic and universal colour domains, places, transitions with their corresponding attributes (e.g., firing-modes, guards) and sub-nets, and the connectivity of the model, i.e., arc with their corresponding functions. This means that other components of the framework can read the files and process the model.

Editor Hardware/Software Requirements

The *CCPetNet-Editor* works with HP 700 Workstations under X-Windows, the base is operating system UNIX HP-UX Release 9-01. The integrated development environment is composed of Windows4GL (/77/) together with the INGRES data base (cpn-db). The editor has to have an access into the file *CCPetNetPICS*, where the pictures of all the buttons in GIF format are. Another file, *IMAGES*, must also be defined in order to receive the pictures for the net information and the sub-net symbols.

6.2 Behavior Analyzer of H-L-PN

The methodology proposed for the development and implementation of H-L-PN-based DECS offers the possibility to analyze a H-L-PN specification before the actual implementation. The engineering-tool presented here includes a set of analysis methods that allow gain confidence on the H-L-PN-based description, as well as detect certain wrongly described specifications, such as incompatibilities and omission errors. Some of the offered functionalities are:

- Syntactic and semantic checking of the ASCII-file generated by the editor.
- Analysis of the H-L-PN itself for structural and behavioral - qualitative - properties.
- Analysis of the interpretation of the nets, i.e., validation of specifications of the modelled structures.
- Quantitative analysis of the H-L-PN for calculating performance indexes of the modelled systems. This is performed using discrete-simulation of a temporized form of the nets (see section 2.5.3).

Comparison of the sorts of Petri Nets used in this work with standard Petri nets descriptions reported currently in the world, results the following main conclusion:

The syntactical and semantical structure of the H-L-PN-based descriptions used here allows performing all the above named analysis methods with commercial CAD systems. These systems relieve the designers of DECS of tedious validation work under some conditions in the H-L-PN theory. These are actually very difficult to proof by using computer-based automatic verification procedures. Some of the world-known CAD tools that can be incorporated to this approach are: *GreatSPN* (/16/), *PETSY* (/58/),

Pascell (/22/), *Design-CPN* /60/, /86/. The fact is that each of these analysis tools can be joined with *CCPetNet* for performing some of the analysis methods addressed above and aiding in model design and debugging. Fig. 58 depicts a modeling-analysis approach combining the H-L-PN-based descriptions generated by *CCPetNet* and the analysis capabilities offered by *Pascell* (Synchronized and Temporized Petri nets) (/22/) and by *Design-CPN* (Colored Petri Nets) /86/.

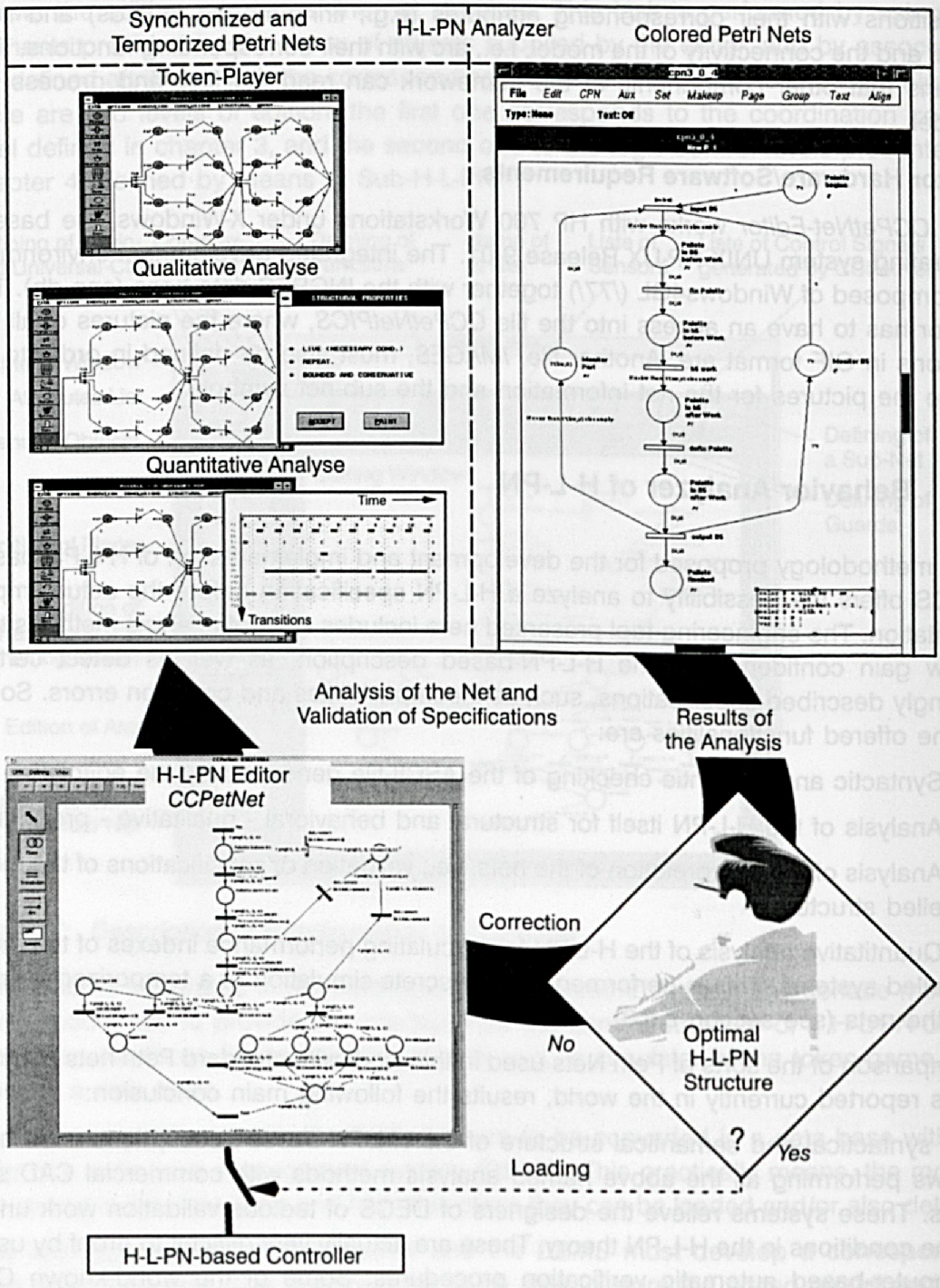


Fig. 58: Integration of H-L-PN Edition and Analysis for Optimizing Control Logic

The proposal of this approach is that the H-L-PN-based description of a FPS and its control logic will be setting into operation - as DECS - if and only if the modeling-analysis approach results an optimal H-L-PN-based control structure from the point of view of the required control specifications.

6.3 Implementation of the H-L-PN Controller into a PC-Platform

Taking into consideration a classification of implementation forms for Petri Net-based real-time controllers presented in /102/, the proposal here is to implement a centralized concurrent control architecture, in a PC-based platform, as depicted in Fig. 59.

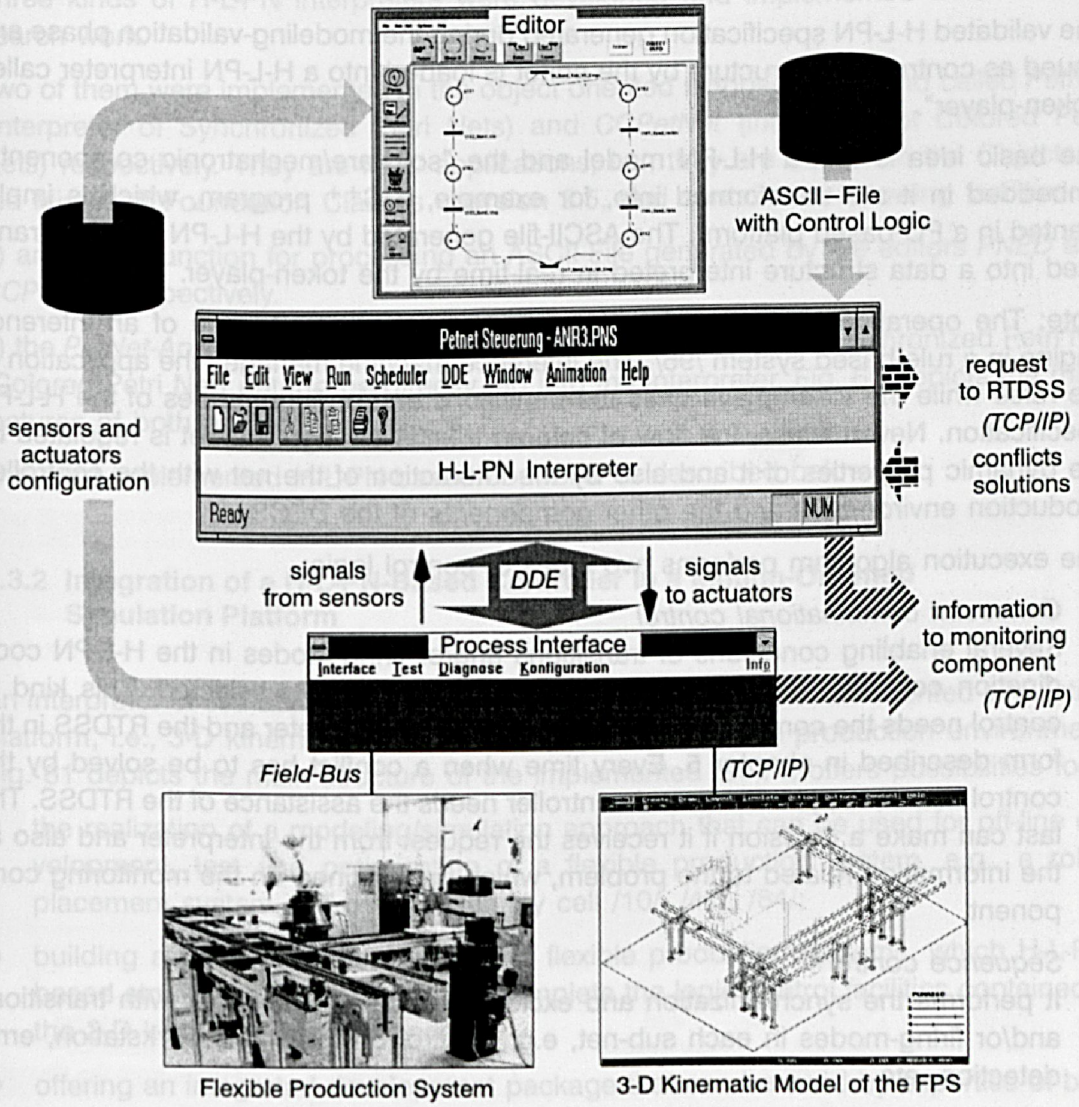


Fig. 59: Implemented PC-based Controller of FPS with H-L-PN

It is composed of:

- many specific tasks and each of them associated with one firing-mode or transition of a H-L-PN,
- a control kernel that evolves according to the enabling/firing rules of H-L-PN described in chapters 2 and 4 and operates on an explicit representation of the net marking. This means that the kernel performs the "token-player" on the net model and in a synchronized form, exchange information and control signals, associated with the firing of transitions and/or firing-modes, with the controlled production environment or a 3-D kinematic simulation model of it.

6.3.1 Structure of the Controller and H-L-PN Execution Algorithm

The validated H-L-PN specification generated during the modeling-validation phase and issued as control logic structure by the editor is loaded into a H-L-PN interpreter called "token-player".

The basic idea is that a H-L-PN model and the "software/mechatronic components" embedded in it are transformed into, for example, a C++ program, which is implemented in a PC-based platform. The ASCII-file generated by the H-L-PN editor is translated into a data structure interpreted in real-time by the token-player.

Note: The operation of the H-L-PN interpreter is very similar to one of an inference engine in a rule-based system /96/. The inference machine manages the application of the rules while the token-player fires the transitions and/or firing-modes of the H-L-PN specification. Nevertheless, the flow of colored tokens through the net is regulated by the dynamic properties of it and also by the interaction of the net with the controlled production environment and the other components of the DECS.

The execution algorithm performs two types of control logic:

- *Constraint combinational control*
  Several enabling conditions of transitions and/or firing-modes in the H-L-PN coordination controller are checked and the firing optimum is selected. This kind of control needs the communication between H-L-PN interpreter and the RTDSS in the form described in chapter 5. Every time when a conflict has to be solved by the control kernel, the H-L-PN-based controller needs the assistance of the RTDSS. The last can make a decision if it receives the request from the interpreter and also all the information related to the problem, which is contained on the monitoring component.
- *Sequence control*
  It performs the synchronization and exclusion of tasks associated with transitions and/or firing-modes in each sub-net, e.g., control on tools in a workstation, error detection, etc.

In order to successfully control the evolution of a production environment, the H-L-PN interpreter, as a principal part of the virtual production environment defined in this chapter, recognizes a set of commands and functions, which are invoked to interpret

the H-L-PN model described in the ASCII-file generated by *CCPetNet* and can also be used:

- to interactively execute the H-L-PN model in real-time conditions,
- to communicate with input-, output-modules of the process interface to initiate appropriate operations through the controlled production environment,
- to display information about some attributes of the net, e.g., marking of places, enabling-condition of firing-modes of transitions, etc., and
- to call routines to handle communications between control kernel and the other components of the hierarchical DECS, i.e., monitoring and visualization, RTDSS, etc.

Three kinds of H-L-PN interpreters were developed and implemented within this research work.

Two of them were implemented in the object oriented language C++ and called *PetNet* (interpreter of Synchronized Petri Nets) and *CCPetNet* (interpreter of Colored Petri Nets) respectively. They are MFC applications, i.e., they are based on the Objects of the Microsoft Foundation Classes - version 2.5, and contain the following facilities:

1) an import function for processing an ASCII-File generated by the editors *PNED* and *CCPetNet* respectively.

2) the *PetNet-Application* (*CCPetNet-Application*) which allows a Synchronized Petri Net (Colored Petri Net) data to be directly fed into the interpreter. Fig. 60 depicts the main features of both named Applications.

The third implemented H-L-PN-based controller is described below.

6.3.2 Integration of a H-L-PN-Based Controller in a Motion-Oriented Simulation Platform

An interpreter of self-adjusting Petri nets is integrated into a motion-oriented simulation platform, i.e., 3-D kinematic simulation, which models a real production environment. Fig. 61 depicts the main structure of the implemented tool. It offers possibilities for

- the realization of a modeling/simulation approach that can be used for off-line development, test and optimization of a flexible production system, e.g., a robot placement system, a flexible assembly cell /10/, /41/, /84/;
- building a discrete-event controller of flexible production systems, which H-L-PN-based structure and functionalities complete the logic control facilities contained in the 3-D kinematic platform; and
- offering an integrated development package that can be used by expertise of both fields (discrete-event and motion-oriented systems) and also can be implemented at industrial level, because of its user-friendly process-visualization and operation facilities.

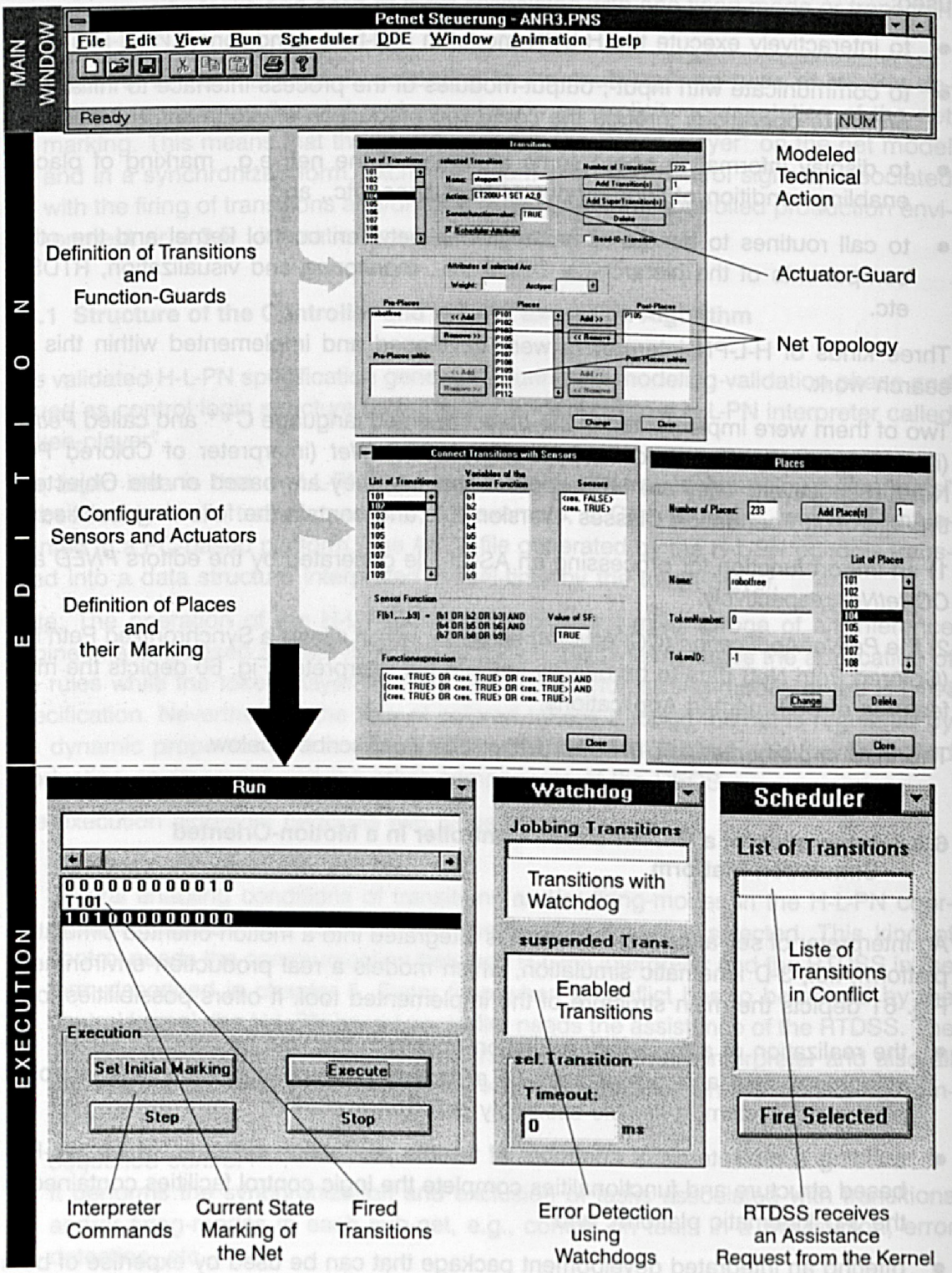


Fig. 60: Main functionalities presented by the PetNet-Application

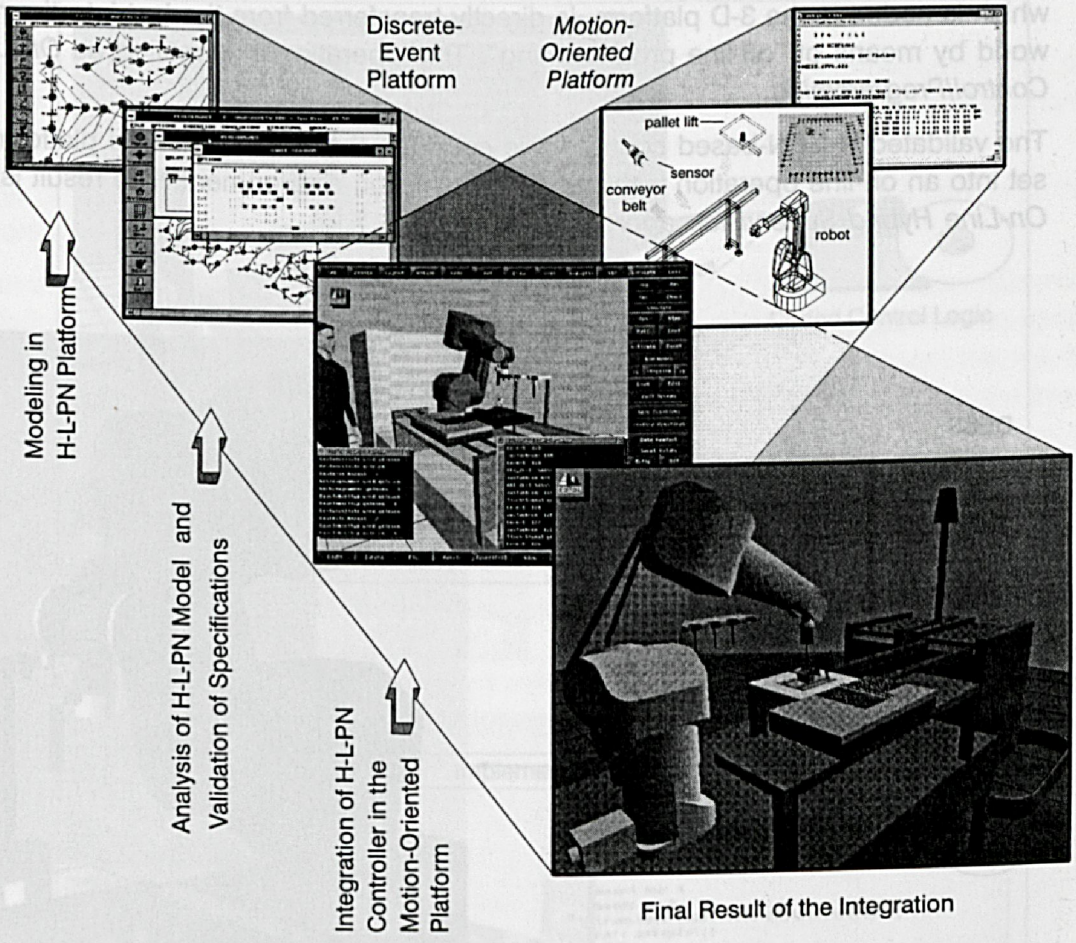


Fig. 61: Implemented H-L-PN Controller in a Motion-Oriented Simulation Platform

Development-, Analysis- and Control-Functionalities of the Implemented Tool

After both systems have been integrated, a flexible production system designer has two platforms, which act as an unique entity. Both systems, i.e., Discrete-Event and Motion-Oriented, exchange messages which allow them to evolve synchronized (see Fig. 62). An off-line simulation-based "verification" of the behavior of the whole structure is then performed. Depending on the results of this off-line simulation, the optimization of control parameters and corrections of errors produced during the development phase, e.g., collisions of devices, identification of restricting areas, are issued in order to get a reliable control logic and safe behavior of the manufacturing system. In this context, the 3-D-based model of the production system replaces the real manufacturing environment.

Only after the control logic embedded in the discrete-event controller and also in the motion-oriented model is tested intensively, it should be transferred to the hardware components of the real flexible production system. In this case, there are two main possibilities for implementing the discrete-event control system:

- The validated control logic embedded in the H-L-PN-based DECS component, which is coded in the 3-D platform, is directly transferred from the last into the real world by means of "off-line programming". This operation is identified as *Off-Line Control/Programming*.
- The validated H-L-PN-based control logic coded in the motion-oriented platform is set into an on-line operation with the real production environment. The result is an *On-Line Hybrid Supervision* approach.

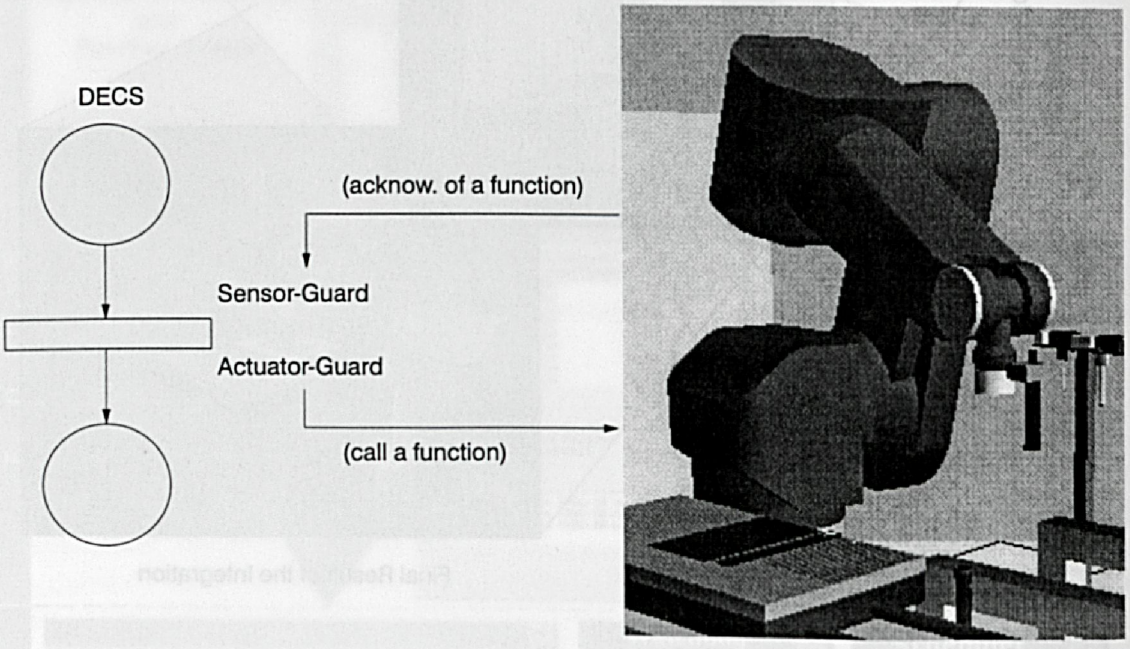


Fig. 62: Exchange of Information between DECS and Motion-Oriented Simulation

Off-Line Control

For the purposes of this work, a program was produced, with ability to translate the H-L-PN-based DEC-program embedded in the motion-oriented simulation platform, e.g., graphic Simulation Language (GSL) (/87/), into another one compatible with the operation language of the real system. Fig. 63 summarizes the main features of the implemented approach.

On-Line Hybrid Supervision

The integrated self-adjusting, synchronized Petri Net and the 3-D kinematic simulation components act as a virtual production system which evolves in a synchronized manner with the real production environment. Fig. 64 shows the main features of the implemented approach.

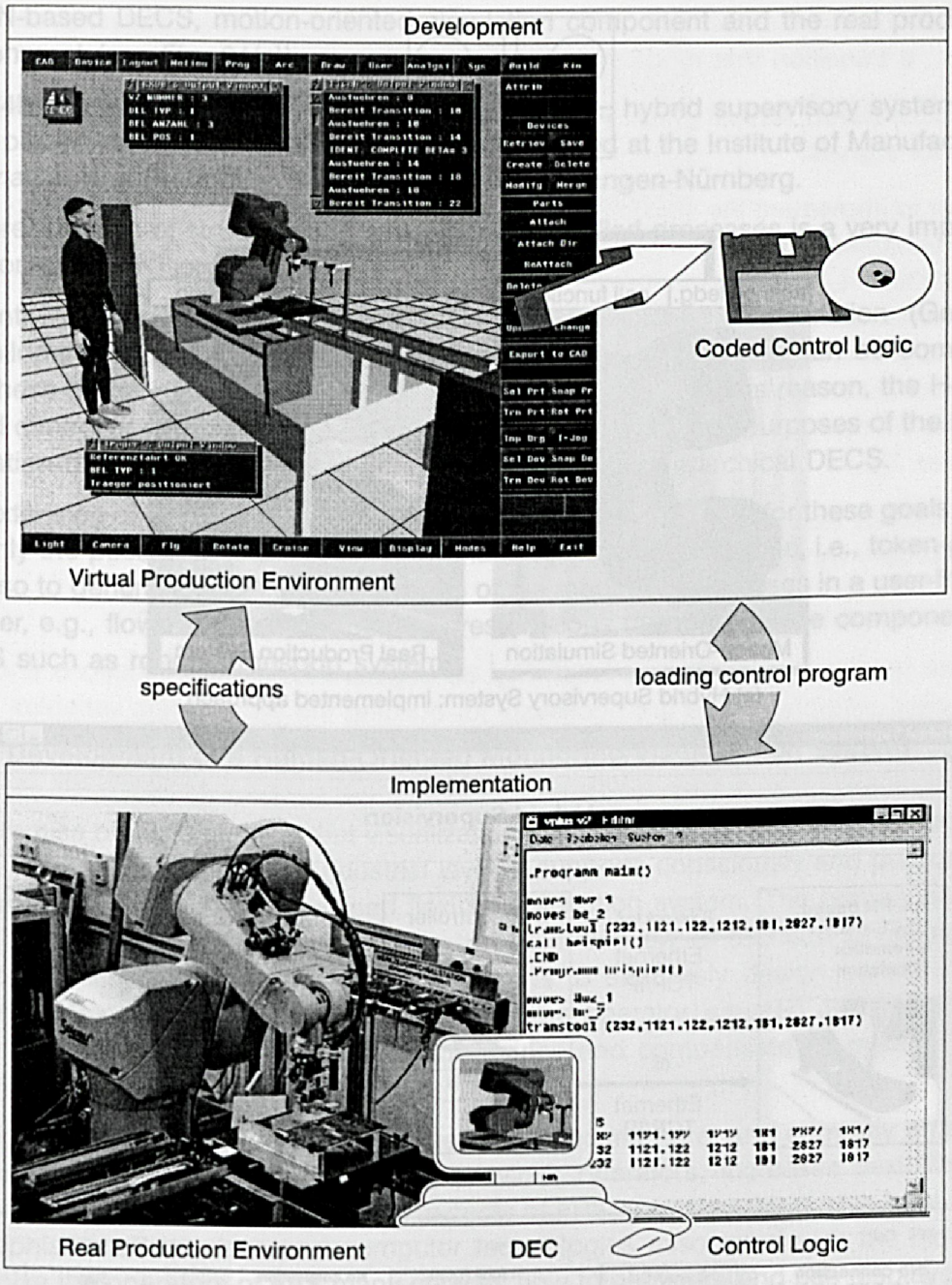
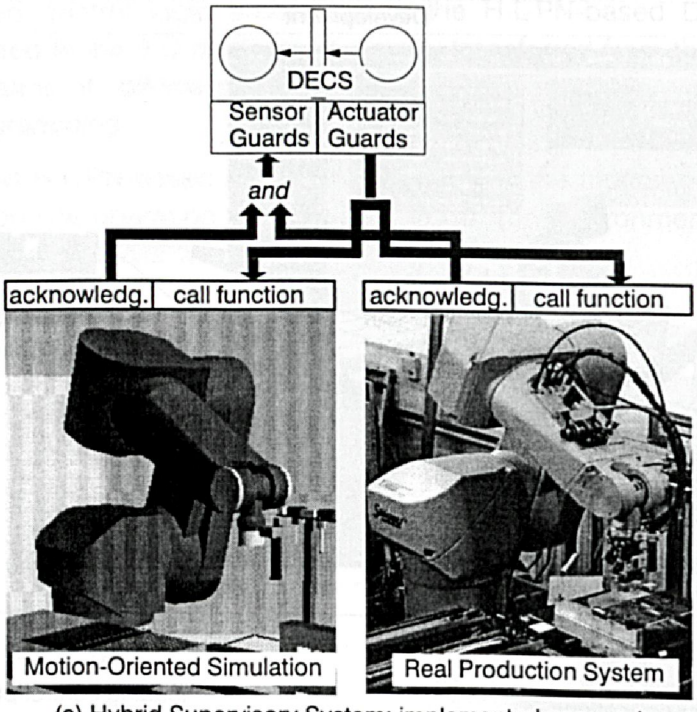


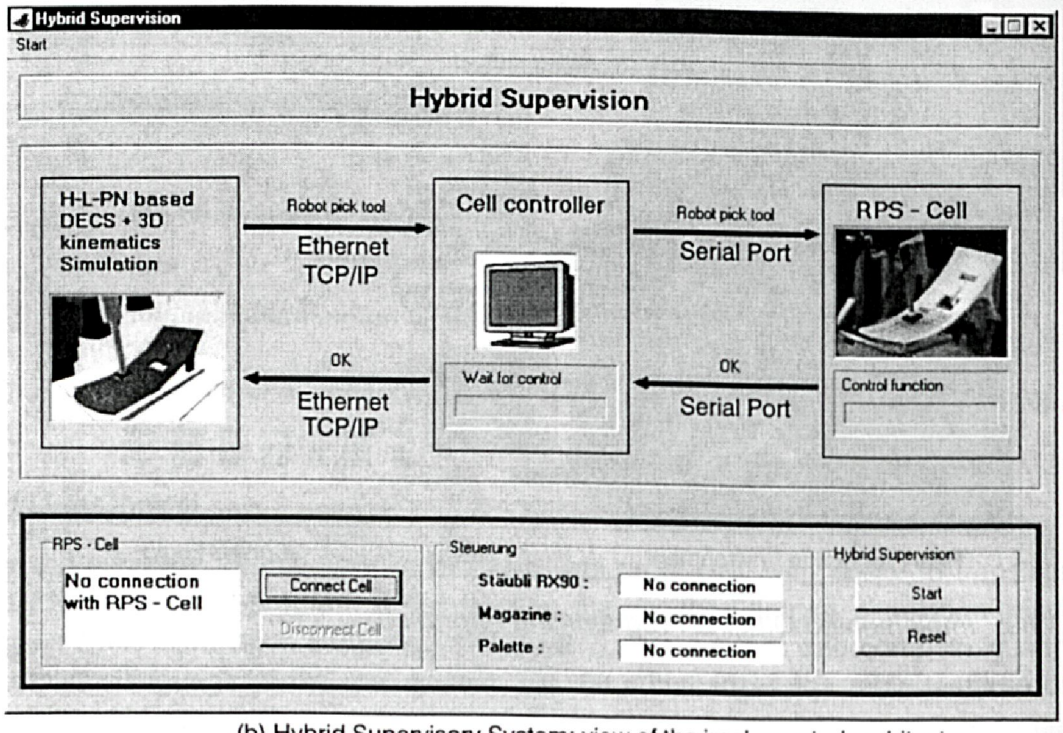
Fig. 63: Implemented Off-Line Programming Approach

6.4 Implementation of a H-L-PN-Based Monitoring/Visualization System using 2-D Visualization Tools

High-Level Petri Net models used in this work are suitable not only for the control of FPS but also for the visualization, and because of their graphical and mathematical character also useful in the field of process monitoring (chapter 4).



(a) Hybrid Supervisory System: implemented approach



(b) Hybrid Supervisory System: view of the implemented architecture

Fig. 64: Integration of H-L-PN and 3-D Kinematic Simulation for On-Line Control

As a matter of fact, the structure and dynamic properties of the H-L-PN-based control logic remain equal, as described in chapter 4, but the evolution of the net will be now performed by taking into consideration the signal, and information, exchange between

H-L-PN-based DECS, motion-oriented simulation component and the real production environment (see Fig. 64(a)).

Fig. 64b shows a view of an experimental setup, i.e., hybrid supervisory system, of a robot placement system (RPS), as it was implemented at the Institute of Manufacturing Automation and Production Systems, University Erlangen-Nürnberg.

The presentation of structure and state of the controlled processes is a very important monitoring/supervisory function.

Currently implemented visualization tools use flow-picture presentation (German: Fließbilddarstellung), which structure and presentation capabilities can be compared with these of the High-Level Petri Nets /3/, /15/, /49/, /53/. For this reason, the H-L-PN-based controller presented in this work can be extended for the purposes of the implementation of a visualization/monitoring component of a hierarchical DECS.

The extension of the functionalities of the H-L-PN-based controller for these goals offers not only the possibility to perform the animation of the nets dynamic, i.e., token-game, but also to generate graphic presentations of the modelled processes in a user-friendly manner, e.g., flow-diagrams or abstract presentations of the hardware components of a FPS such as robots, transport systems.

6.4.1 Development of a Human-Oriented Monitoring/Visualization System

The premise of this section is that visualization is a critical issue for the implementation of the H-L-PN-based DECS at industrial level. It supports consciously and purposefully human interaction with a supervised flexible production system. The human operator plays a big role in FPS, specially from the point of view of monitoring/visualization functions /65/. For this reason, the challenge is to effectively design human-system interaction, here: the H-L-PN-based DECS and an operator, situated at the workshop level, in a way that enhance human performance and compensate for system limitations.

Putting together H-L-PN-based DECS like one proposed here, and currently emerging industrial monitoring/visualization tools, provides operators, increases flexibility and widens a range of choice in almost all aspects of FPS operation. Main idea here is that the sophisticated interface and computer technologies associated with the industrial tools give the operators of FPS many choices how to configure and use displays, controls, etc. together with the potentialities offered by a H-L-PN-based DECS structure like the one presented in this work.

For the purposes of the implementation of the monitoring/visualization component of the DECS in a FPS-operator-friendly manner, new presentation methods were observed and formed with taking into consideration ergonomic concepts in the context of new technology within the machine-human-interface theory (MHI).

Two main sources of information were considered in order to perform the monitoring/visualization of the processes, states of hardware components, mechanical and electrical parameters of a FPS, etc:

- model-based statical and dynamical parameters of the H-L-PN-based controller, e.g., a transition that models an operation is enabled to fire;
- feature-based statical parameters located at the process interface, e.g., sensor/actuator signals.

In order to implement the visualization component of the DECS it is necessary to define all the variables related to hardware components of the flexible production system, and also of the H-L-PN-based controller, related with the information addressed above. All these variables are recognized in this work as a *set of process variables X*. It is also necessary to implement an interface between the monitored production processes and the visualization processes. The interface is responsible for a reliable data transmission and the mapping from processes variables to visualization variables (*set of internal variables Y*).

This concept can be represented as a mathematical function *f*, which is performed automatically by the monitoring system.

$$X \text{ (process variables)} \rightarrow \quad f \quad \rightarrow \quad Y \text{ (internal variables)}$$

In order to help ground this concept, Fig. 65 depicts the Boolean functions and trees of process variables which are necessary for building the visualization variables "*pallet movement from buffer 1 to buffer 2*" and "*pallet movement from buffer 4 to buffer 2*" in a sector of a sample flexible assembly cell, controlled and monitored with the proposed H-L-PN-based approach (Note: More details about this example can be found in /39/).

Elements of the set of internal variable *Y* have to be connected with different graphic objects in order to perform a dynamic monitoring/visualization of all controlled processes. The generation of the graphic objects and the codification of all information is performed in an object oriented form, according to the norm /29/. Furthermore, the flow-picture-based monitoring/visualization is converted into textual information in order to make the work of the human operator easier, according to the norm /114/.

6.4.2 Combining H-L-PN-Based Information and Technical Signals for Monitoring/ Visualization Tasks

For the operators of a FPS is the information, coming from H-L-PN-based controller and from process interface, very difficult to understand, because of its form (without structure). For this reason, the processing of this information and its representation in an operator-friendly manner are the main tasks to be performed for building of a monitoring/visualization component for the usage on industrial level.

The aim of Fig. 66 is to explain the above named concept.

The implemented monitoring/visualization component consists of three presentation levels: a) the level of hardware components, b) logic control level and c) general view level.

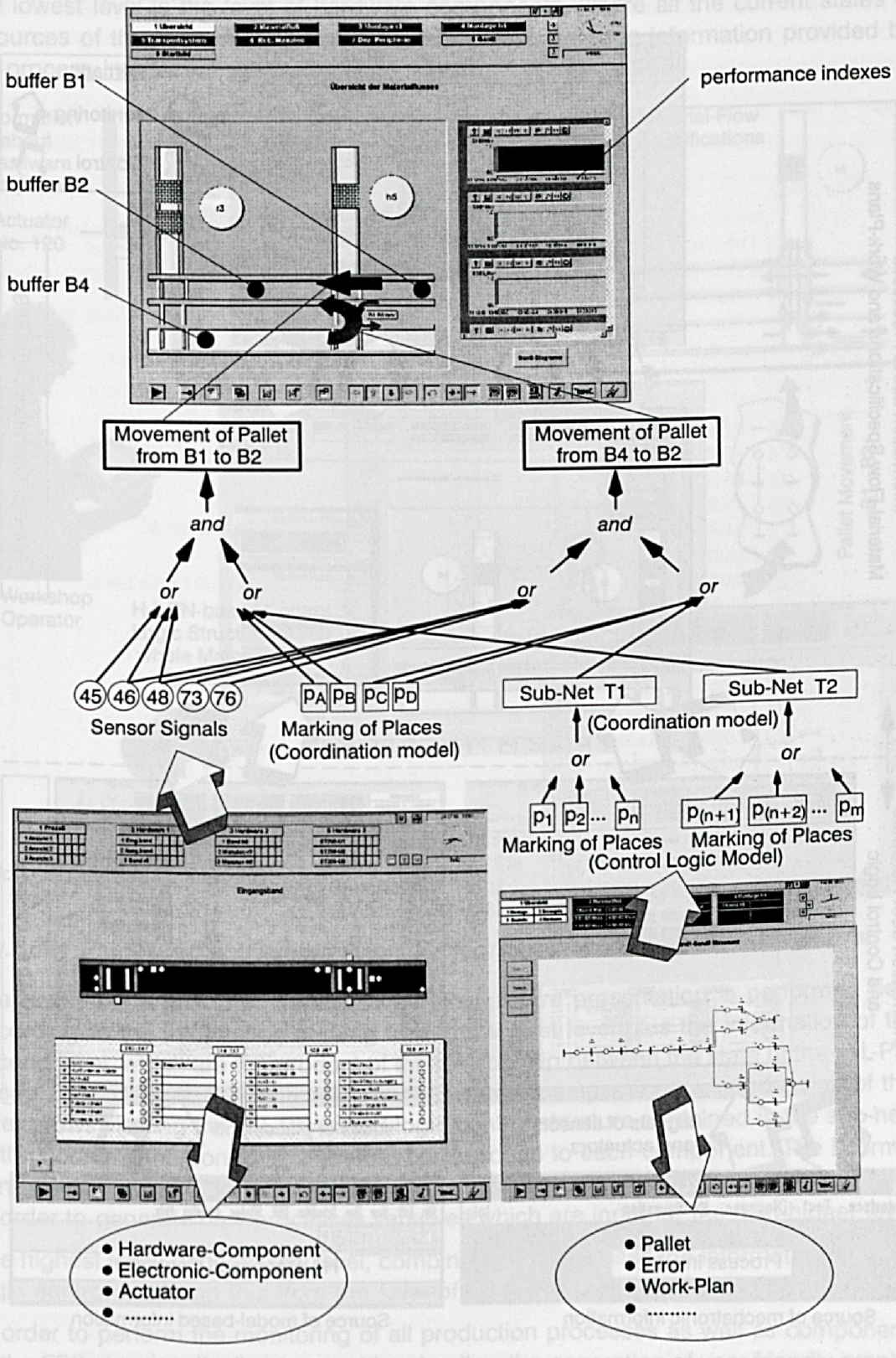


Fig. 65: Algorithm for converting of Process Variables into Visualization Variables

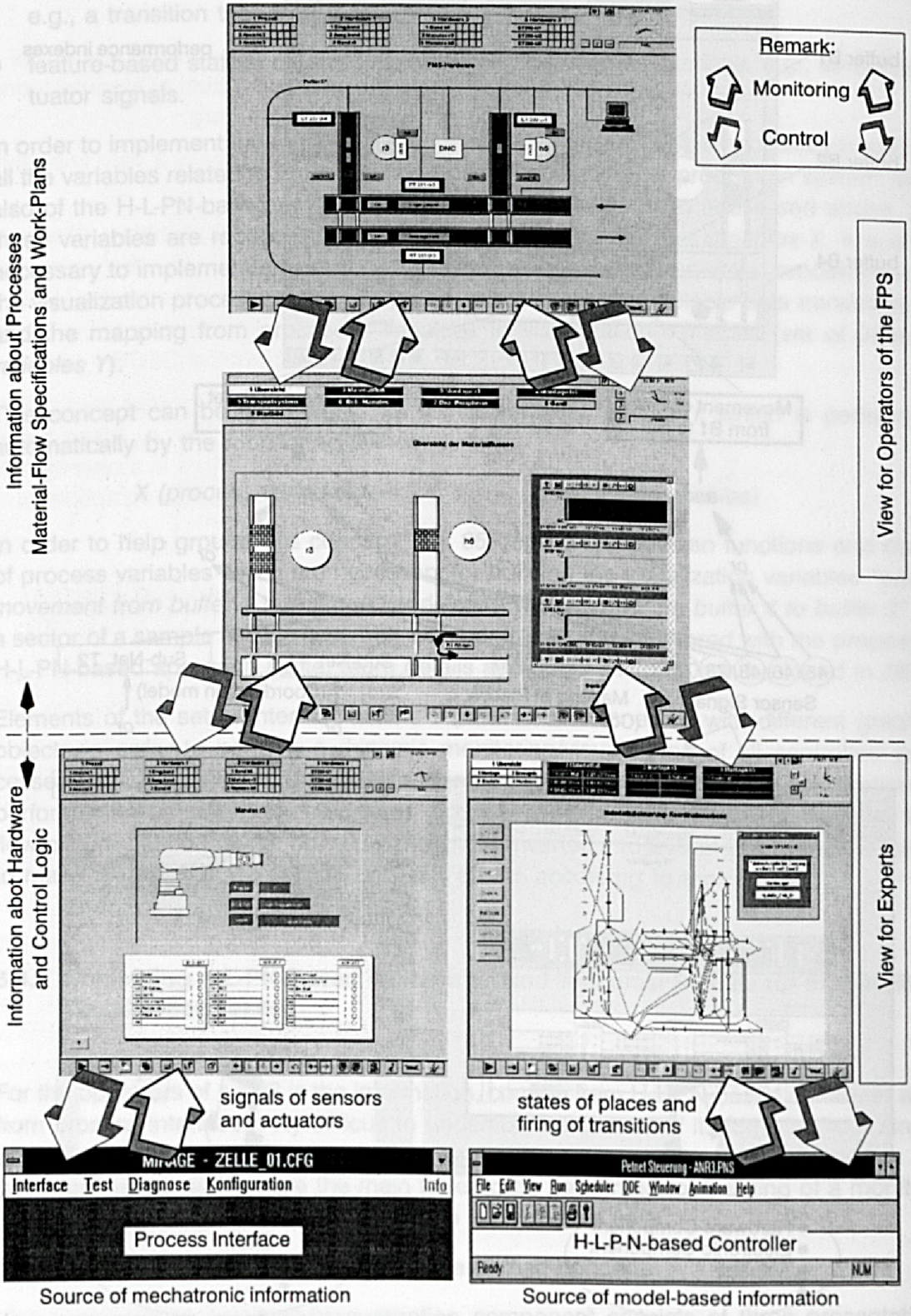


Fig. 66: Hierarchical Structure of the implemented Monitoring/Visualization System

The lowest level is the level of hardware components, where all the current states of resources of the system are visualized, considering only the information provided by the process interface.

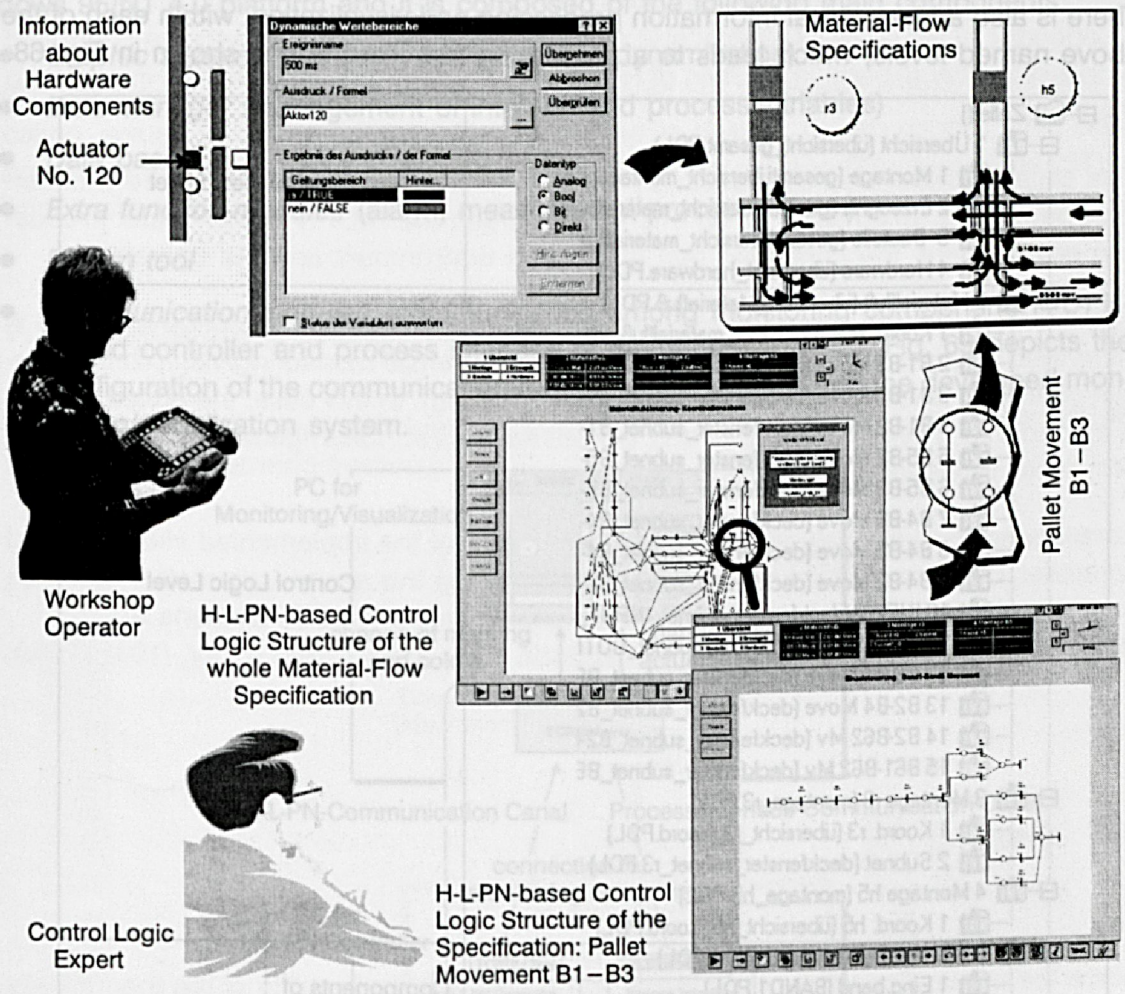


Fig. 67: Processing and Interpretation of Information within each Monitoring-Level

The next level is the logic control level. Flow-picture presentation is performed here according to the information coming from the lowest level plus the information of the second level, which are the product of the relationship between the state of the H-L-PN-based controller and the material-flow and work-plans specifications. Extension of this level allows viewing the H-L-PN-based logic control structure contained in the sub-nets of the coordination control system that corresponds to each component. The information obtained at this level is processed according to the algorithm depicted in Fig. 65 in order to generate new processes variables which are input for the next higher level.

The highest level, general view level, combines and processes the information from the logic control level and this from the level of hardware components.

In order to perform the monitoring of all production processes as well as components of the FPS, the visualization concept has to allow the generation of user-friendly graphic presentation of all this information in each level. All process variables generated at each

of the lower levels have to be converted with the help of Boolean-Algebra into new information, i.e., visualization variables. This results in an ergonomical flow-picture presentation of the whole production process at the highest level.

There is also a hierarchical information processing and visualization, within each of the above named levels, which leads to a processing-tree like the one shown in Fig. 68.

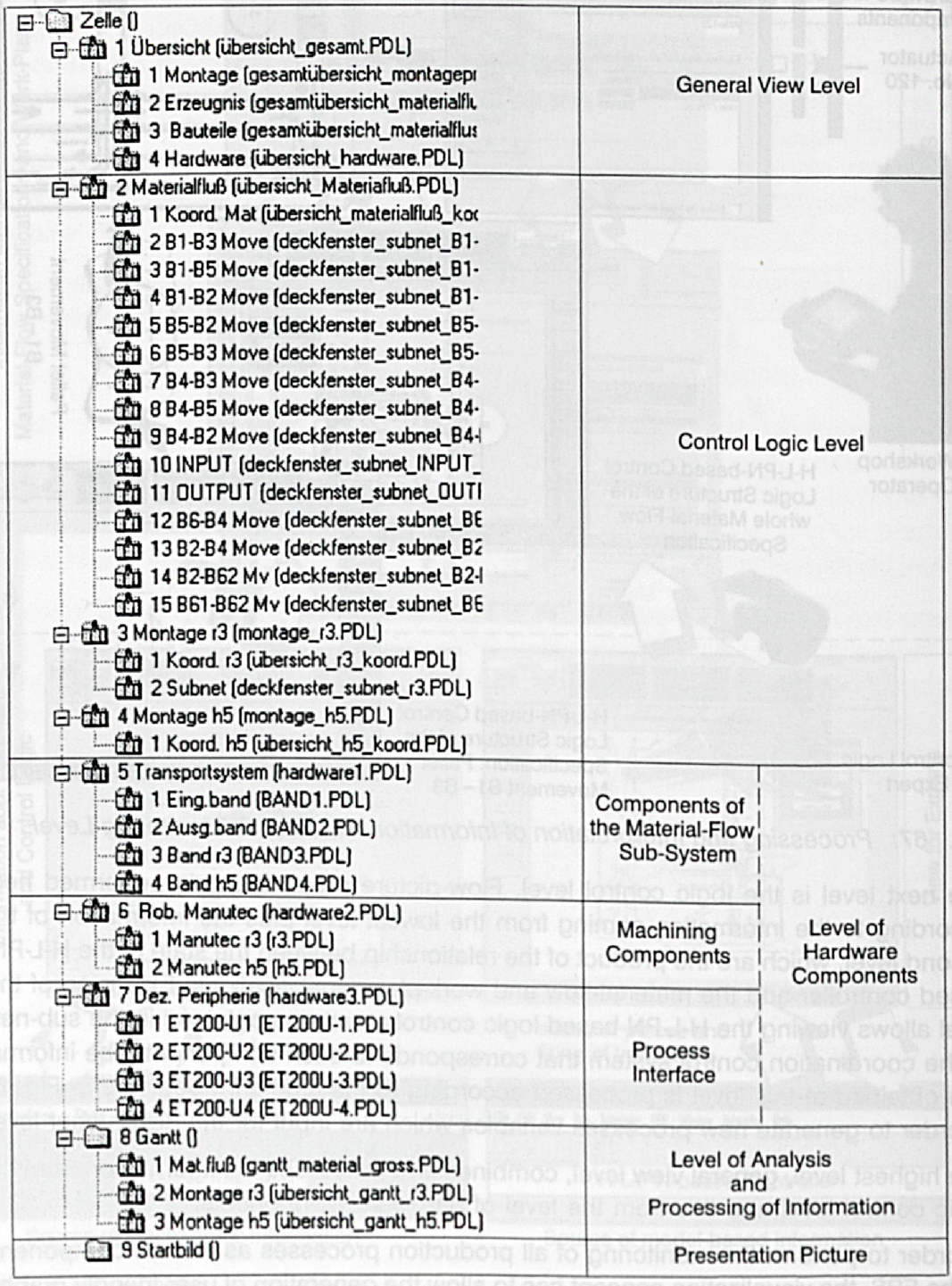


Fig. 68: Hierarchical Processing of Monitoring- and Visualization-Information

6.4.3 Implemented Monitoring/Visualization Component

The software structure of the monitoring/visualization system is implemented in Windows 95/NT 4.0 platform and it is composed of the following main components:

- *Graphic system* (graphic presentation and dynamization)
- *Data manager* (management of internal- and process-variables)
- *Data base*
- *Extra function-modules* (alarm, measurement processes, etc.)
- *Edition tool*
- *Communication-modules* (communication among monitoring component, H-L-PN-based controller and process interface of the controlled FPS). Fig. 69 depicts the configuration of the communication-modules implemented into the developed monitoring/visualization system.

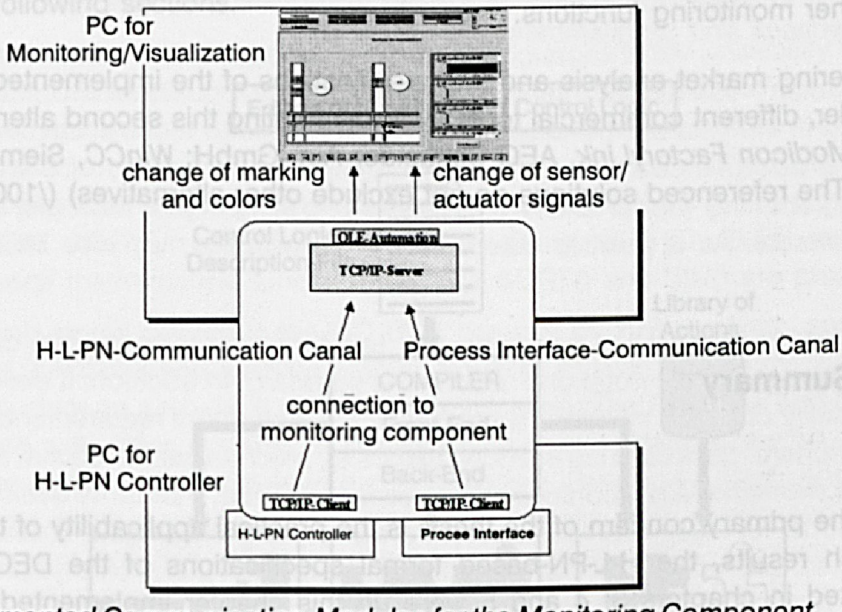


Fig. 69: Implemented Communication-Modules for the Monitoring Component

Two different implementations of the monitoring/visualization component were performed.

1) Development within the H-L-PN-based controller of a monitoring component with utilities that have to be formed as software modules taking into consideration the theory presented in chapters 4 and 5.

In order to visualize the dynamic behavior of the net, an information-flow between the H-L-PN and an animation component can be also realized through the data base coupling referenced in section 6.1. The current H-L-PN condition-data saved in the data base are put at animator's disposal by the editor.

The implementation of such a visualization component was performed for research purposes, within the editor *PNED.*

The results of this implementation have shown that this presentation is not suitable for industrial purposes, because it is not operator-friendly. The problem is that an appropriate trade-off between functionality and understanding of the monitoring/visualization component needs to be made. The more compact a H-L-PN, the more difficult it is for an operator to visualize its statical and dynamical properties related with the controlled processes. Only experts in the field of H-L-PN theory do not have difficulties interpreting the animation.

This situation motivated the necessary development of an industrial-oriented monitoring component, which acts as an interface between the H-L-PN-based virtual production environment, the controlled real production environment and the operators.

2) In order to fill the above mentioned gaps, the animation of the H-L-PN model is integrated as a monitoring function into a 2-D commercial monitoring/visualization system (German: Bedienungs- & Beobachtungswerkzeug). In this case, the edited H-L-PN is loaded into a visualization tool and the animation, i.e., token-game, is synchronized with other monitoring functions.

Considering market analysis and the specifications of the implemented H-L-PN-based controller, different commercial tools for implementing this second alternative could be used: *Modicon FactoryLink*, AEG Schneider Aut. GmbH; *WinCC*, Siemens AG.
(Note: The referenced solutions do not exclude other alternatives) (/100/, /103/, /105/).

6.5 Summary

Since the primary concern of the thesis is the practical applicability of the theoretically research results, the H-L-PN-based formal specifications of the DECS-components presented in chapters 3, 4 and 5 were un this chapter implemented. The following forms of setting an H-L-PN-based DECS into operation were describe:

- implementation of an H-L-PN interpreter in a PC-based platform able to run in a synchronized manner with a controlled and monitored flexible production system;
- integration of the H-L-PN interpreter in a 3-D kinematic platform and generation of control code for FPS in an off-line manner;
- integration of the H-L-PN interpreter in a 3-D kinematic platform and synchronization of both systems with a real production environment in an on-line manner.

Finally, the development and implementation of a H-L-PN-based monitoring/visualization component using a 2-D industrial visualization tool was described, and the results of its implementation at industrial level were shown with examples.

7 Automatic Generation of PLC Code from a H-L-PN-Based Specification of Control Logic

In this chapter, an engineering tool (hardware-platform-independent) is pursued. This was developed to automatically convert a validated H-L-PN-based control logic specification, issued by the graphical editor described in chapter 6, into control logic code according to the international standard IEC 1131-3 – language Structured Text (ST) – and to set it into operation on an industrial Programmable Logic Controller (PLC).

7.1 Engineering Tool for Generating of IEC 1131 Control Logic Code

Fig. 70 depicts the main structure of the engineering tool, which components are described in the following sections.

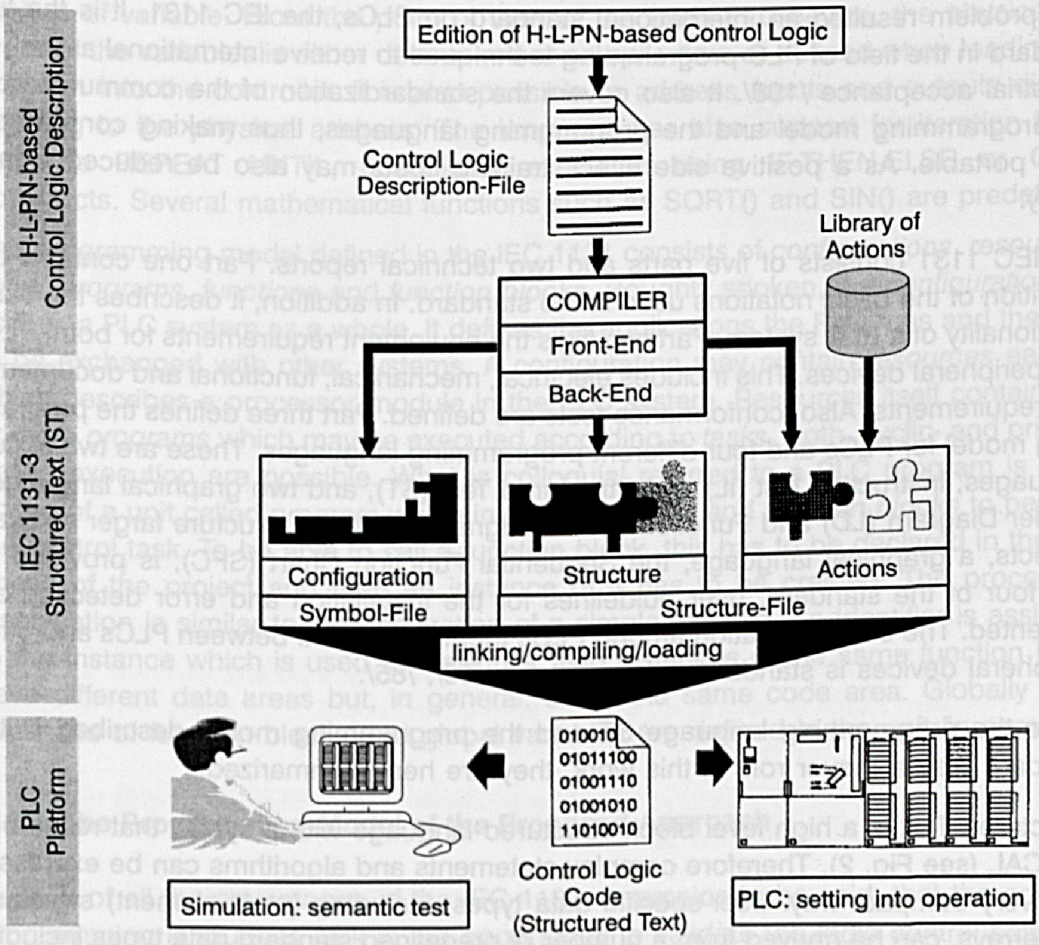


Fig. 70: Engineering Tool to Generate PLC-Code from a H-L-PN-based Specification

7.1.1 Why the International Standard IEC 1131

As stated in chapter 2, one of the main weaknesses of currently implemented DECS are based not on the hardware of the flexible production systems but on common description methods, as well as programming and implementation techniques for the components of the control structure. These differ, among others, in terms of modelling and analysis/validation power, agility, flexibility, clarity.

Moreover, industrial flexible production environments incorporate control equipment from various vendors which collaborate via hardware/software solutions. In order to support the efficient setup and reconfiguration of the manufacturing process as a whole, the interpretability and reusability of hardware and software components has to be guaranteed.

Existing software and hardware solutions in the application area of logic controllers often lack portability. Products from different vendors often have different communication interfaces and programming environments.

This problem resulted an international standard on PLCs, the IEC 1131. It is the first standard in the field of PLC-programming techniques to receive international as well as industrial acceptance /106/. It also covers the standardization of the communication, the programming model and the programming languages, thus making control software portable. As a positive side effect, training costs may also be reduced significantly.

The IEC 1131 consists of five parts and two technical reports. Part one contains the definition of the basic notations used in the standard. In addition, it describes the basic functionality of a PLC system. Part two fixes the equipment requirements for both, PLCs and peripheral devices. This includes electrical, mechanical, functional and documentation requirements. Also, conformance tests are defined. Part three defines the programming model for PLCs and four different programming languages. These are two textual languages, Instruction List (IL) and Structured Text (ST), and two graphical languages, Ladder Diagram (LD) and Function Block Diagram (FBD). To structure larger software projects, a graphical language, the Sequential Function Chart (SFC), is provided. In part four of the standard, user guidelines for the installation and error detection are presented. The communication among PLCs themselves, and between PLCs and other peripheral devices is standardized in part five /56/, /85/.

Since the programming language ST and the programming model described in the standard play a mayor role in this work, they are here summarized.

Structured Text is a high level block structured language with a syntax that resembles PASCAL (see Fig. 2). Therefore complex statements and algorithms can be expressed in a very compact way. User-specific data types, including (multi-element) structures and arrays, can be derived from a number of predefined standard data types including BOOL, INT, WORD and DWORD. There are also specific predefined data types for the management of digital and analogue values, times-of-day, dates and durations.

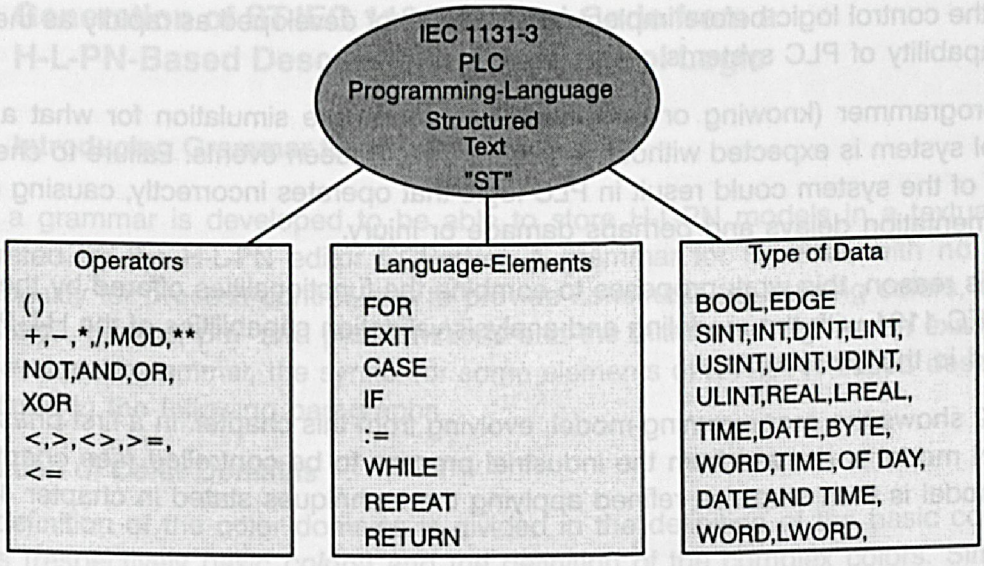


Fig. 71: Main Characteristics of the ST-IEC 1131 Programming Language

A contribution to modern PLC-based programming languages is the possibility for declaration of variable. From the point of view of the used PLC-device, the physical address of the variables is either determined at compile-time or defined when loading the program into the controller. It is also possible to address inputs and outputs directly referring to the physical address. The language has also support for iteration loops such as REPEAT UNTIL, conditional execution using IF-THEN-ELSE or CASE constructs. Several mathematical functions such as SQRT() and SIN() are predefined.

The programming model defined in the IEC 1131 consists of *configurations*, *resources*, *tasks*, *programs*, *functions* and *function blocks*. Roughly spoken, the *configuration* describes a PLC system as a whole. It defines the applications the PLC runs and the data to be exchanged with other systems. A configuration may contain *resources* each of which describes a processor module in the PLC system. Resources itself contain one or more *programs* which may be executed according to *tasks*. Both, cyclic- and priority-based execution are possible. What is colloquial referred to a PLC program is composed of a unit called *program* which invokes *functions* and *function blocks* to perform the control task. To be able to call a function block, this has to be declared in the first phase of the project and then an instance of it has to be created. The process of instantiation is similar to the declaration of a simple variable. An identifier is assigned to the instance which is used for calling it. Two instances of the same function block have different data areas but, in general, share the same code area. Globally used instances of function blocks and global variables are defined in the unit "program".

7.1.2 The Programming-Model of the Proposed Approach

Despite of all the advantages of the IEC 1131, it remains to be said, that the covered programming languages lack in the opportunity to validate the logic control software. Only simulation can be used in order to check both the fulfillment of the specification and the correctness of the software beforehand. The ability to verify, validate and docu-

ment the control logic before implementation has not developed as rapidly as the control capability of PLC systems.

The programmer (knowing or not) tends to perform the simulation for what a logic control system is expected without considering unforeseen events. Failure to check all states of the system could result in PLC logic that operates incorrectly, causing costly implementation delays and perhaps damage or injury.

For this reason, this work proposes to combine the functionalities offered by the standard IEC 1131 with the modeling and analysis/validation capabilities of the H-L-PN described in the last chapters.

Fig. 72 shows the programming-model, evolving from this chapter. In a first phase, the H-L-PN model is derived from the industrial process to be controlled (see chapter 3). The model is then stepwise refined applying the techniques stated in chapter 4.

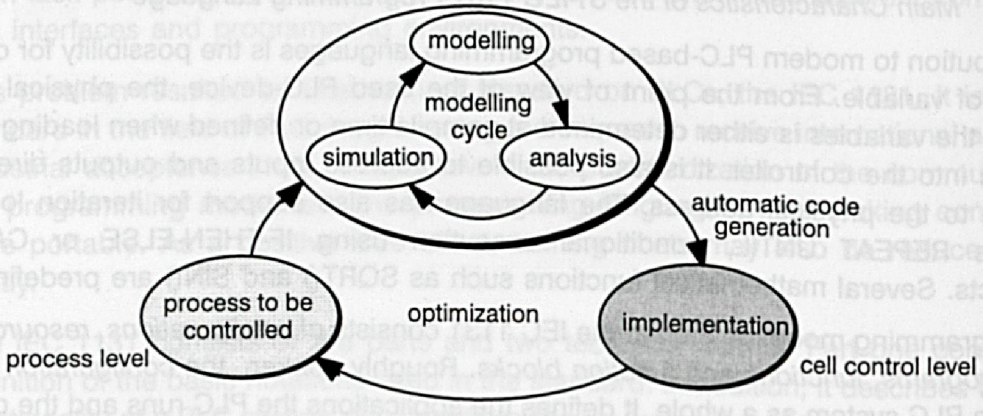


Fig. 72: Programming-Model used in this Work

The implementation phase is automated by a compiler generating a PLC program from the H-L-PN model. If necessary, the modeler portions the H-L-PN model and distributes it among different PLCs. Each part is then compiled separately. The communication and synchronization among the PLCs is subject to the modeler at the moment. IEC 1131 programming tools available on the market, like /78/, /79/, /80/, /81/, /82/, are used to create functions and function blocks that are dynamically invoked within actions. They also assist in setting up the run time environment. Principally, two modelling techniques can be distinguished when implementing actions:

- Simple actions, i.e., the setting of actuator values, are expressed by the terms provided by the grammar (see section 7.2.1).
- Functions or function blocks are used to perform complex actions. Then, the modeler has to write code in one of the IEC 1131 languages, which is invoked from the H-L-PN model. The code is called within a scan cycle of the PLC or simulator, and has to be short.

7.2 Generation of ST-IEC 1131 Control Code from a H-L-PN-Based Description of the Control Logic

7.2.1 Introducing Grammar for H-L-PN

First, a grammar is developed to be able to store H-L-PN models in a textual form generated by the H-L-PN editor *CCPetNet*. A grammar for H-L-PNs, with no added functionality for process control, has to provide constructs for defining colors, places, transitions, guards, pre- and post-functions and the initial marking. As an example of the developed grammar, the syntax for some elements of a H-L-PN-based description is defined in the following paragraphs.

Definition of Color Domains

The definition of the color domains is divided in the definition of the basic color domains (respectively *basic colors*) and the definition of the complex colors. Similar to a variable name in a programming language, a color name is assigned to each color. A basic color is defined by its name and an integer interval to express the possible color tones. The integer interval starts from number one to a user-defined upper bound. In Fig. 73(a) there are three different basic color domains specified, namely A, B and C. Complex colors are the Cartesian product of basic colors. They are defined by its name and the enumeration of the corresponding basic colors (see Fig. 73(b)).

```
define_basic_colors
  color A = integer [1 to 10]
  color B = integer [1 to 13]
  color C = integer [1 to 30]
end_colors
```

(a)

```
define_complex_colors
  color D = product A * B
  color E = product A * C
  color F = product B * C
end_complex_colors
```

(b)

```
define_places
  place p1 = color A, maxtoken 2
  place p2 = color E, maxtoken 20
  place p3 = color universal, maxtoken 20
  place p4 = color discolored, maxtoken 13
  place p5 = color universal, maxtoken 22
end_places
```

(c)

```
define_transitions
  transition t1 from p3 to p5
  transition t2 from (p2,p3,p4) to (p1,p5)
end_transitions
```

(d)

```
define_marking
  place p1 = (1) + (2)
  place p2 = (4,2)
  place p3 = (1,2,1) + 4*(2,2,2)
  place p4 = discolored (5)
end_marking
```

(e)

Fig. 73: (a) Definition of Basic Colors, (b) Definition of Complex Colors, (c) Definition of Transitions, (d) Definition of Places, (e) Definition of the Initial Marking

Definition of Places and Transitions

A place is defined by its name, the associated color set and an integer value indicating how many tokens of the specified color set can at most reside in the place. The last value is needed for the implementation of the compiler. It is obtained from the validation

phase of the H-L-PN-based control system (see chapter 3). The universal color type is specified by the keyword *universal*, the discolored color by the keyword *discolored*. An example is given in Fig. 73(c).

A transition is defined by its name followed by the list of input places and the list of output places (see Fig. 73(d)).

Definition of the Initial Marking

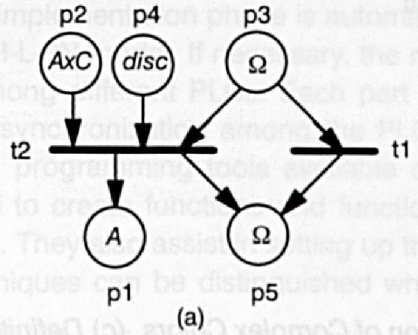
The initial marking of a place is defined in the form of a list, with the list elements separated by the symbol "+". Each list element describes a set of tokens that reside in the place. A single token is described by an expression enclosed in curved brackets like "(1)" or "(1,2,1)". The values enclosed in the brackets define the color tones of the token corresponding to the color set of the place. Place *p1* in Fig. 73(e) has the color set A which is a basic color. The expression "(8)" describes therefore a token of the color set A with color tone "8".

To define multiple occurrences of one token, a multiplicative factor can be added as a prefix. An example is given with the initialization expression "4*(2,2,2)" of place *p3* denoting four occurrences of the token "(2,2,2)". Places which color set is the discolored color are initialized with the keyword *discolored*, followed by the number of discolored tokens (see place *p4*).

Definition of Functions

Fig. 74 illustrates how the arc functions (i.e., pre- and post-function) are defined. Therefore, Fig. 74(a) gives a graphical representation of the transitions of Fig. 73(d). The example shown in Fig. 74(b) refers to this graph.

Multiple occurrences of an element of the multi-set are defined by an integer value followed by the prefix symbol "*". The elements of the multi-set are denoted in the form of a list separated by the symbol "+". The composition of functions is denoted with the symbol "#".




```
define_pre
  pre (t2,p4) = 3*abs
  pre (t2,p3) = 2*id
  pre (t1,p3) = succ(2)#pred(3) + id
  pre (t2,p2) = 2*proj(1,3)#succ(1)#pred(3)
end_pre

define_post
  post (t2,p1) = proj(1)
  post (t2,p5) = succ(1) + succ(2)
  post (t1,p5) = succ(3) + pred(1)
end_pre
```

(b)

Fig. 74: Graphical Representation of the Transitions of Fig. 73(e) and Definition of Pre- and Post-Functions within the Grammar

Definition of Guards

According to Definitions 13, the guard function *G(t)* of a transition *t* is defined in disjunctive normal form:

$$G(t) = \vee_i \, G\&_i(t), \text{ with } G\&_i(t) = \wedge_j \, g_{ij}(t).$$

The term $g_{ij}(t)$ denotes a comparison expression where integer values are compared using the operators "=,<,>,<=,>=". The integer values are either obtained as a result of a projection function (keyword *proj*) applied to a token or they are integer constants. This is illustrated in Fig. 75(a). The guard for transition *t1* is fulfilled in a marking *M(p)* if there is a token in *M(p)* which second component has value "*10*" or which second component is one number less than its third component. The symbol "#" denotes the composition of functions.

Definition 40 introduced the dispatcher-guard *DF(t)*. Within the grammar, the dispatcher-guard is incorporated in the guard. A special attribute within the guard is used to mark the colored tokens that might cause a conflict at run time. Fig. 75(b) illustrates this. Principally, transition *t2* is enabled only for tokens which second component have the value "*1*", "*2*" or "*3*". Conflicts can arise from tokens which second component have the value "*1*" or "2". They are marked with the keyword *scheduled* which is by the compiler mapped on the corresponding bdv_i-variables (see definition 40). The RTDSS is in this case asked at run time to decide, according to a predefined strategy or algorithm, whether *t2* is allowed to fire or not (see chapter 5).

```
define_guards
  guard t1 = proj(2)=10 or
             proj(2)=proj(3)#succ(3)
end_guards
```

(a)

```
define_guards
  guard t2 = scheduled and proj(2)=1 or
  scheduled and proj(2)=2 or  proj(2)=3
end_guards
```

(b)

Fig. 75: (a) The Guard Function of a Transition, (b) The Guard may contain a Special Attribute evaluated by the Scheduler (RTDSS)

7.2.2 Extensions of the Grammar for Modelling Logic Controllers

In the following paragraphs, some of the extensions presented in chapter 4 are incorporated in the grammar. As explained there, the incorporation of the sensor/actuator interface is done by the testing of sensor values in the sensor-guards *SG(t)* and the setting of actuator values within actions associated with transitions or firing-modes, i.e., action-guards *AG(t)*, at the level of the sub-nets. The set of sensor-signals and actuator-signals is implemented by a set of *I/O variables*. They are similar to variables in programming languages and used to address physical I/O locations.

Definition of I/O variables

Physical I/O locations of a PLC are addressed via I/O variables which are defined according to the IEC 1131 standard. Optionally, initialization values can be assigned. An example is given in Fig. 76(a). The variables *movex* and *movey* are of type word and initialized with zero. Variable *emergency_off* is of type Boolean and variable *status* is of type double word. The physical location is specified after the symbol '%'. This is done in IEC 1131 syntax.

```
define_iec_io_variables
  movex at %QW1 : WORD := 0
  movey at %QW20 : WORD := 0
  emergency_off at %QX3 : BOOL
  status at %ID1 : DWORD
end_iec_io_variables
```

(a)

```
define_io_guards
  io_guard t1 = status > 10 and movex = 1
  io_guard t2 = movey <= 5 or not emer-
gency_off = false
end_io_guards
```

(b)

Fig. 76: (a) Definition of I/O Variables. (b) Definition of I/O Guards.

Sensor-guards are Boolean expressions over the I/O variables. Their definition is exemplarily shown in Fig. 76(b).

Definition of Agents and Actions

According to the modeling method presented in chapter 3, an operation in the real production environment, e.g., loading a pallet, is modeled by means of a transition or firing-mode in the H-L-PN coordination model. If this transition or firing-mode becomes enabled, the control logic has to be transferred to a sub-net (see chapter 4).

The grammar provides a construction for binding both control levels: the *agent,* which concept and the object-oriented concept are well connected. The development of the agent-concept shows following aspects (see Fig. 77):

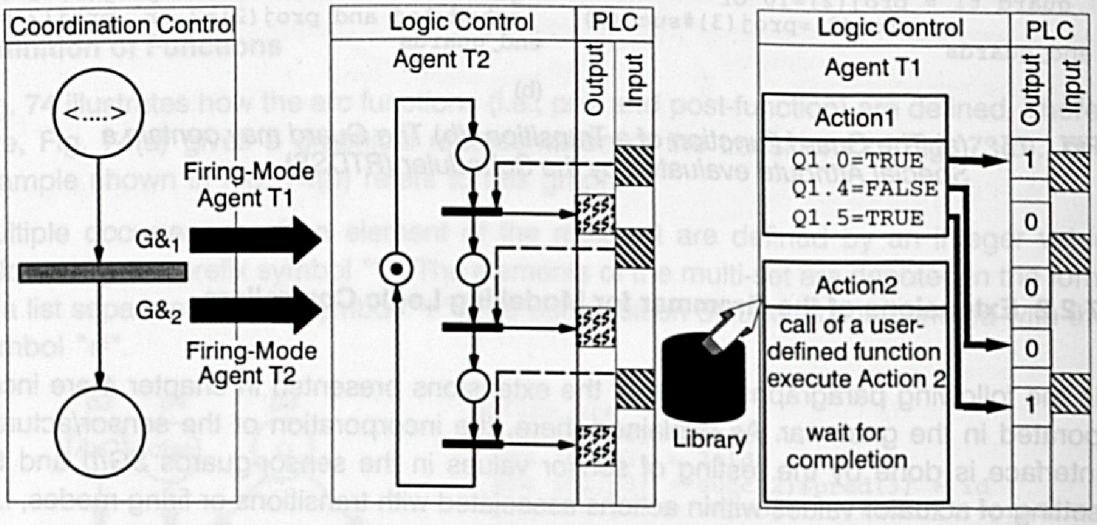


Fig. 77: Developed Agent-Concept

- An agent is associated with a transition or firing-mode in the coordination model, i.e., a sub-net in the logic control level, and its function is to call the set of actions associated with the transitions of the sub-net (see Fig. 78(a)). Within an instruction scope, positioned between DEFINE_AGENT and END_AGENT, can be defined single transitions for the agents.
- Every agent defines 1...n control actions.

The control actions that have to be enforced from sub-nets are intended to set actuator values depending on input/output values, i.e., signals from sensors and actuators associated with sensor- and action-guards, and/or the current enabled firing-mode (see

chapter 4). Within the grammar, this is formulated by *if-then-else* statements (Fig. 78 (b)).

The agents control the start of the actions, which are called with the key word CALL, but they do not implement them.

- The direct implementation of input- and output-variables within an agent is not allowed. Only actions may have direct access to memory of the PLC.
- The called actions can themselves start other actions.

According to the task, there are three alternatives to implement the called actions. They can be implemented as:

- library action, which have to be declared within the grammar construction "include",
- user-defined action, which have to be also declared within the grammar construction "include", or
- inline action, which have to be be declared within the grammar construction "DEFINE_ACTION" and "END_ACTION".

Because the implementation of the control actions happens in a separate grammar construct, for the agent it is not important to which of three named sorts the action belongs.

Inline actions are implemented within the description data. The implementation of the library actions and user-defined actions will take place later. Both will be declared in the quell data of the compiler.

```
define_io_variables
  band4 : BOOL
  stopper4 : BOOL
  bewegenx : WORD
  ende at %Q1.0 : BOOL
end_io_variables

define_agents
  agent transition t1 = {
    if (proj(2)#succ(2)) then
      ende = FALSE
      bewegenx = 10
    else if (proj(3) = 2) then
      ende = TRUE
      bewegenx = 15
    else
      ende = FALSE
      call (p_laden)
    endif
  }
```

(a)

```
include_actions
  p_laden : FB86
  p_bewegen : transport
end_include

define_actions
  action entladen = {
    band4 = TRUE;
    stopper4 = FALSE;
  }
end_actions
```

(b)

Fig. 78: (a) Definition of Agents, (b) Definition of Actions

The interface provided by the grammar for calling functions and function blocks is according to the IEC 1131 standard and it allows calling functions defined in one of the

IEC 1131 languages and stored in a library. The application context for this facility is wide-spread.

- Complex operations can be implemented within functions or function blocks thus hiding the implementation details. This may be specific algorithms, e.g., for motion control or communication.
- Existing IEC 1131 software modules and standardized function blocks like "time modules" can be easily linked to the H-L-PN model.

An application example is presented in Fig 79. A barcode-reader reads the barcode of oncoming items. This is done in the function block instance *read_barcode*. The return value *returnval* of this function block instance is stored in the component *barcode* of the actual firetoken for later evaluation. The read-operation is mapped on the transition *Read* which is enabled only if a new item is available. This condition is tested with an sensor guard and a Boolean I/O variable *new_item* indicating new oncoming items.

7.2.3 Program for Compiling ST-IEC 1131-3 Control Logic Code according to the H-L-PN Descriptions

Notations

Based upon the international standard IEC 1131 described in section 7.1.1, the following notations are used in this section. A *PLC application* is considered to consist of a *run time environment*, that is one configuration, one or more resource and one or more task, and a *PLC program*. The latter consists of the unit "program" together with functions and function blocks. It is assumed, that a PLC application or PLC program runs on one PLC.

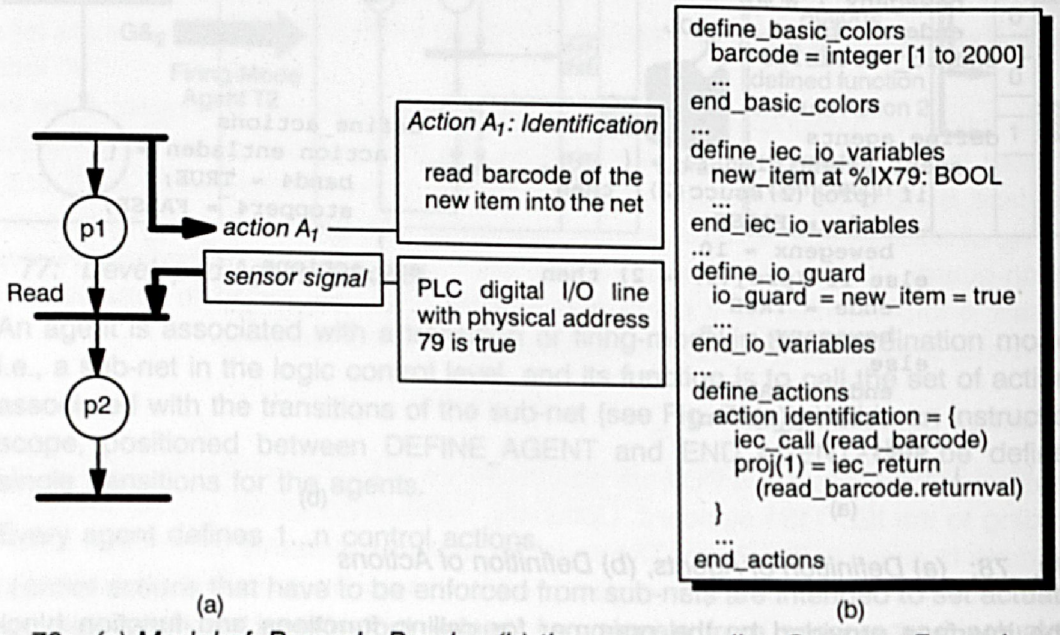


Fig. 79: (a) Model of Barcode-Reader, (b) the corresponding Grammar Example

Choosing a Target Language

The compiler in question (CONCORDE: *Compiler for Generating IEC 1131 PLC-Code from High-Level Petri-Nets*) generates from a given textual H-L-PN description-file a logic controller program in the IEC 1131-3 language Structured Text (ST). Structured Text is the chosen target language for the compiler, because it is a high level language and can be perfectly adapted to the form of information contained in the H-L-PN description to be compiled. Thus, complex algorithms can be expressed concisely and short, making the generated code readable and easier to validate and maintain. Also, the niveau difference between the source and target language is minimized compared to all other IEC 1131 languages.

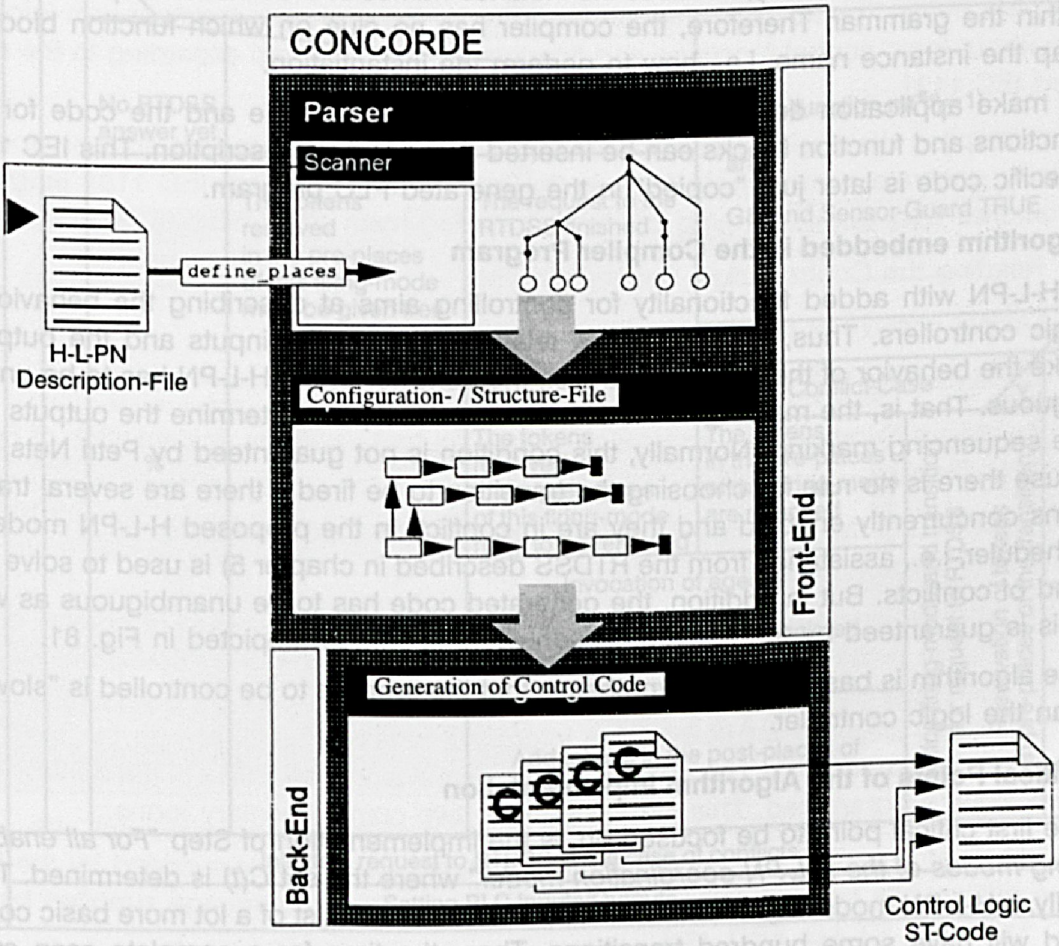


Fig. 80: Main Structure of the Implemented Software "CONCORDE"

Generation of a PLC Application

The desired aim would be to design a compiler that a) automatically portions parts of the H-L-PN model, according to production steps, to different PLCs and generates the necessary communication functions and b) generates for each part the whole PLC application.

State of the art is b) (see Fig. 80). However not the whole PLC application, but a PLC program is generated by the compiler. The reason is a loss of flexibility, arising if additional programs shall run on the same PLC. Because this results in changes of the automatically generated configuration and tasks. Therefore, the approach pursued here is to generate only the PLC program and use market-available, standardized tools for setting up the run time environment.

Normally, the textual description of the H-L-PN model is sufficient for generating the PLC program. However, if the call interface is used to call functions or function blocks, then a) the coding of these functions and function blocks has to be done separately and b) the instantiation of the function blocks has to be done separately. The latter can not be done automatically, because the instance names of the function blocks are used within the grammar. Therefore, the compiler has no clue on which function block to map the instance name, i.e., how to perform the instantiation.

To make application development easy, the instantiation code and the code for the functions and function blocks can be inserted in the textual description. This IEC 1131 specific code is later just "copied" in the generated PLC program.

Algorithm embedded in the Compiler Program

A H-L-PN with added functionality for controlling aims at describing the behavior of logic controllers. Thus, it specifies the relation between the inputs and the outputs. Alike the behavior of the logical controller, the behavior of the H-L-PN has to be unambiguous. That is, the marking of the H-L-PN and the inputs determine the outputs and the sequencing marking. Normally, this condition is not guaranteed by Petri Nets, because there is no rule for choosing the transition to be fired if there are several transitions concurrently enabled and they are in conflict. In the proposed H-L-PN model, a scheduler, i.e., assistance from the RTDSS described in chapter 5) is used to solve this kind of conflicts. But in addition, the generated code has to be unambiguous as well. This is guaranteed by the implementation of the algorithm depicted in Fig. 81.

The algorithm is based on the hypotheses that the process to be controlled is "slower" than the logic controller.

Critical Points of the Algorithm Implementation

The first critical point to be focused on, is the implementation of Step *"For all enabled firing-modes of the H-L-PN coordination model"* where the set *C(t)* is determined. Typically, a H-L-PN modelling an industrial application will consist of a lot more basic colors and will have some hundred transitions. Then, the time for a complete scan cycle would no longer meet the requirements in the process industries.

The solution is a) to scale down the set of possible firing-modes for a transition (i.e., *C(t)*) and b) to portion a complex industrial application on more than one PLC, thus reducing the total number of transitions for a given PLC. Part a) is implemented by taking into account the tokens in the pre-places of a transition when computing the set *C(t)*. Because *de facto*, the tokens in the pre-places of a transition determine the possible firing-modes. It can be shown, that in the most common case, when not all pre-functions of a transition *t* are projection functions, the cardinality of the set *C(t)* is limited

by the least capacity of the pre-places of *t*. Otherwise, the limit depends on the number and kind of projection functions as well as on the capacity of the pre-places. Part b) is at the moment subject to the modeler (see section 7.1.2).

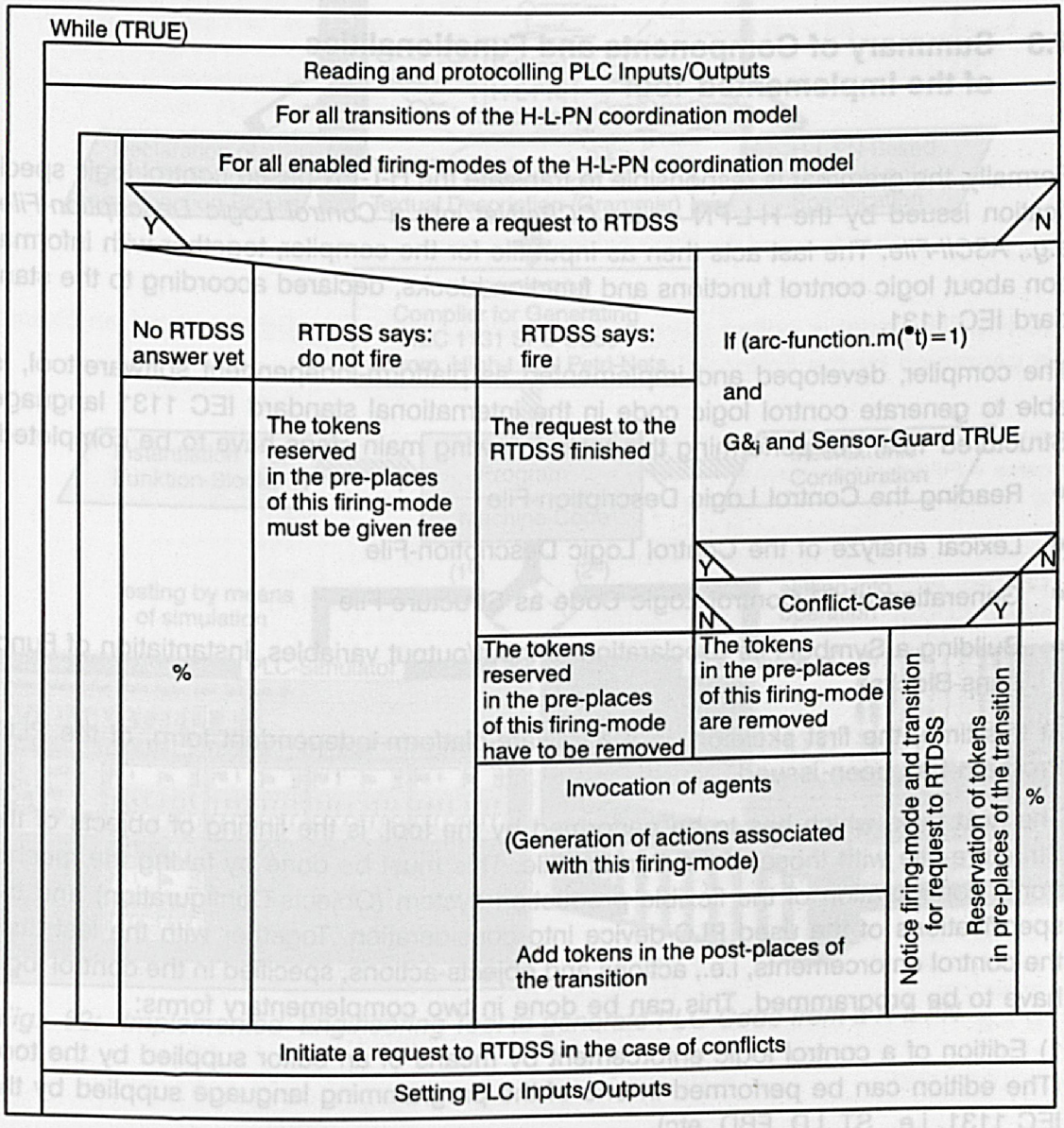


Fig. 81: Algorithm implemented in the CONCORDE-Compiler

A second critical aspect is that the controller program has to allocate memory to store the tokens residing in a place. However, none of the IEC 1131 languages does support dynamic storage allocation. Thus, an upper bound for the number of tokens that can be stored in each place has to be identified. This value is the capacity of the place and is obtained from the validation phase of the H-L-PN-based control system (see chapters 3 and 4).

The last aspect to be discussed is the invocation of agents. As explained above, the scan cycle has to be short and therefore the user-defined code must be short and quickly to execute.

7.3 Summary of Components and Functionalities of the Implemented Tool

Formally, the grammar is responsible to translate the H-L-PN-based control logic specification issued by the H-L-PN editor *CCPetNet* into a *Control Logic Description-File, e.g., ASCII-File*. The last acts then as input file for the compiler, together with information about logic control functions and function-blocks, declared according to the standard IEC 1131.

The compiler, developed and implemented as platform-independent software-tool, is able to generate control logic code in the international standard IEC 1131 language Structured Text. For performing this task, following main steps have to be completed:

- Reading the Control Logic Description-File
- Lexical analyze of the Control Logic Description-File
- Generation of ST-Control Logic Code as Structure-File
- Building a Symbol-File (Declaration of input/output variables, Instantiation of Functions-Blocks)

At this time, the first skeleton, in a hardware-platform-independent form, of the PLC-Program has been issued.

The next task, which has to be performed by the tool, is the linking of objects of the Structure-File with those of the Symbol-File. This must be done by taking the mechatronic configuration of the flexible production system (Objects-Configuration) and the specifications of the used PLC-device into consideration. Together with the last task, the control enforcements, i.e., actions and objects-actions, specified in the control logic have to be programmed. This can be done in two complementary forms:

1) Edition of a control logic enforcement by means of an editor supplied by the tool. (The edition can be performed in one of the programming language supplied by the IEC 1131, i.e., ST, LD, FBD, etc).

2) Call of programmed Objects-Actions, which are stored in a library

This set of steps results a complete ST-IEC 1131 Control Logic Code for the current application (see Fig. 82), which can be

- loaded into an industrial simulation-tool for PLC-Programmes /51/, /83/. This allows to proof the semantic of the generated control logic, according to the specifications of the used PLC-device and the designed H-L-PN-based control logic.
- transferred into the memory of the PLC-device and set into operation.

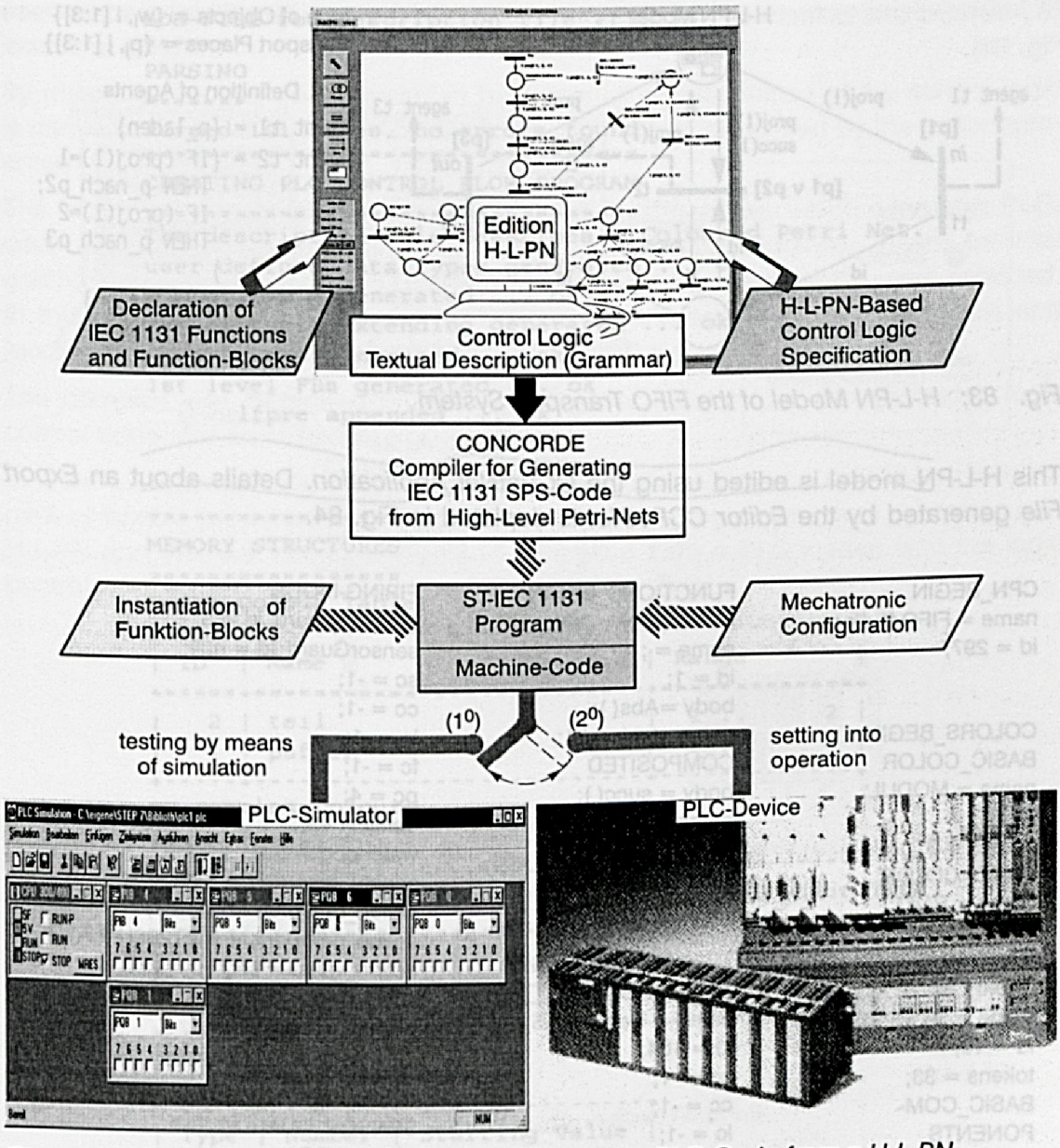


Fig. 82: Implemented Engineering Tool to generate PLC-Code from a H-L-PN

Example of Application

The functioning of the engineering tool is described with the help of an example.

The controlled system is a linear conveyor, i.e., a FIFO Transport System (First-Input-First-Output) with three positions p_j, which can simultaneously transport 3 types of objects w_i. Fig. 83 depicts the H-L-PN-based model of the conveyor which characteristics and specifications are described for example in /25/. A part of the grammar concerning the definition of the agents related with the operations performed in the component is also described.

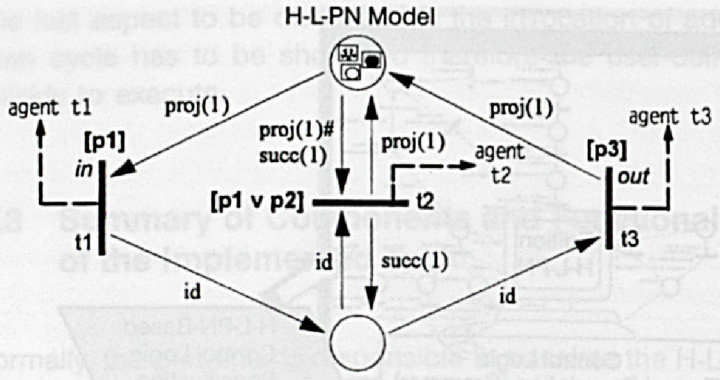


Fig. 83: H-L-PN Model of the FIFO Transport System

This H-L-PN model is edited using the *CCPetNet Application.* Details about an *Export File* generated by the *Editor CCPetNet* is depicted in Fig. 84.

```
CPN_BEGIN
name = FIFO Buffer;
id = 297;

COLORS_BEGIN
BASIC_COLOR
name = MODUL;
id = 2;
tokens = 3;
BASIC_COLOR
name = PART;
id = 4;
tokens = 8;
COMPLEX_COLOR
name = MODPAL;
id = 10;
tokens = 33;
BASIC_COM-
PONENTS
id = 2;
id = 8;
COLORS_END

FUNCTIONS_BEGIN
FUNCTION
name = ;
id = 1;
body =Abs( );
parameter1 = 21;
COMPOSITED
body = succ( );
parameter1 = 2
parameter2 = 1
FUNCTIONS_END

TRANSITIONS_BEGIN
TRANSITION
name = Beladen h5;
id = 104;
sc = -1;
cc = -1;
lc = -1;
fc = -1;
pc = -1;
is_super = 1;
super_id = -1;

FIRING-MODE
functionGuard_id = 3;
sensorGuard_id = null;
sc = -1;
cc = -1;
lc = -1;
fc = -1;
pc = 4;
delay = 43;
watchdog = null;
sensor_resource = S48;
action_resource = Band 1;
action_component = ET200U-1 A1.7;
action_value = set;
subcpn_id = 308;
TRANSITIONS_END

GUARDS_BEGIN
SENSOR_GUARD
name = SG1;
id = 1;
term = Sensor 1
FUNCTION_GUARD
name = FGM1;
id = 1;
term = proj( 1 )=1
GUARDS_END
```

Fig. 84: Export File generated by the Editor CCPetNet

The Export File obtained form the Editor *CCPetNet* is loaded into the compiler *CONCORDE*, which generates a *Log-File* and a protocol of the *ST-IEC 1131* code-generation process like the one depicted in Fig. 85.

```
LOG-FILE for description file <fifo3>
=======
PARSING
=======
Parsed 101 lines, no errors found.
*********************************
CREATING PLC CONTROL FLOW PROGRAM
*********************************
The description file describes a Coloured Petri Net.
user defined data types generated ... ok
symbol table generated ... ok
symbol table extending generated ... ok
Data Blocks generated ... ok
1st level FBs generated ... ok
     lfpre appended ... ok
```

```
==================
MEMORY STRUCTURES
==================
Basic Colour Table:
+-----+-----------------------------+-------------+
| ID  | Name                        | Range       |
+=====+=============================+=============+
|   2 | teil                        |  1 ..     2 |
|   1 | puffer                      |  1 ..     3 |
+-----+-----------------------------+-------------+
no complex colours
Place Table:
+-----+-----------------------------+-----+------------
| ID  | Name                        | Max | Colour Type
+=====+=============================+=====+============
```

```
==============
GENERATED POUs
==============
+-------+--------+----------------+
| Type  | Number | Starting Value |
+=======+========+================+
| UDT   |      5 |      1 (set)   |
| DB    |     11 |     10 (set)   |
| FC    |      2 |     41 (set)   |
| FB    |     78 |     10 (set)   |
| OB    |      2 |        ---     |
| I/O   |      9 |        ---     |
+-------+--------+----------------+
            ****** END LOG-FILE ******
```

Fig. 85: Log-File generated by the Compiler CONCORDE

The ST-IEC 1131 code is loaded into a programming system that generates the *control-code*. It is loaded into the PLC, which communicates with the hardware components,

i.e., sensors and actuators, of the FIFO system by means of a Field-Bus interface (see Fig. 86).

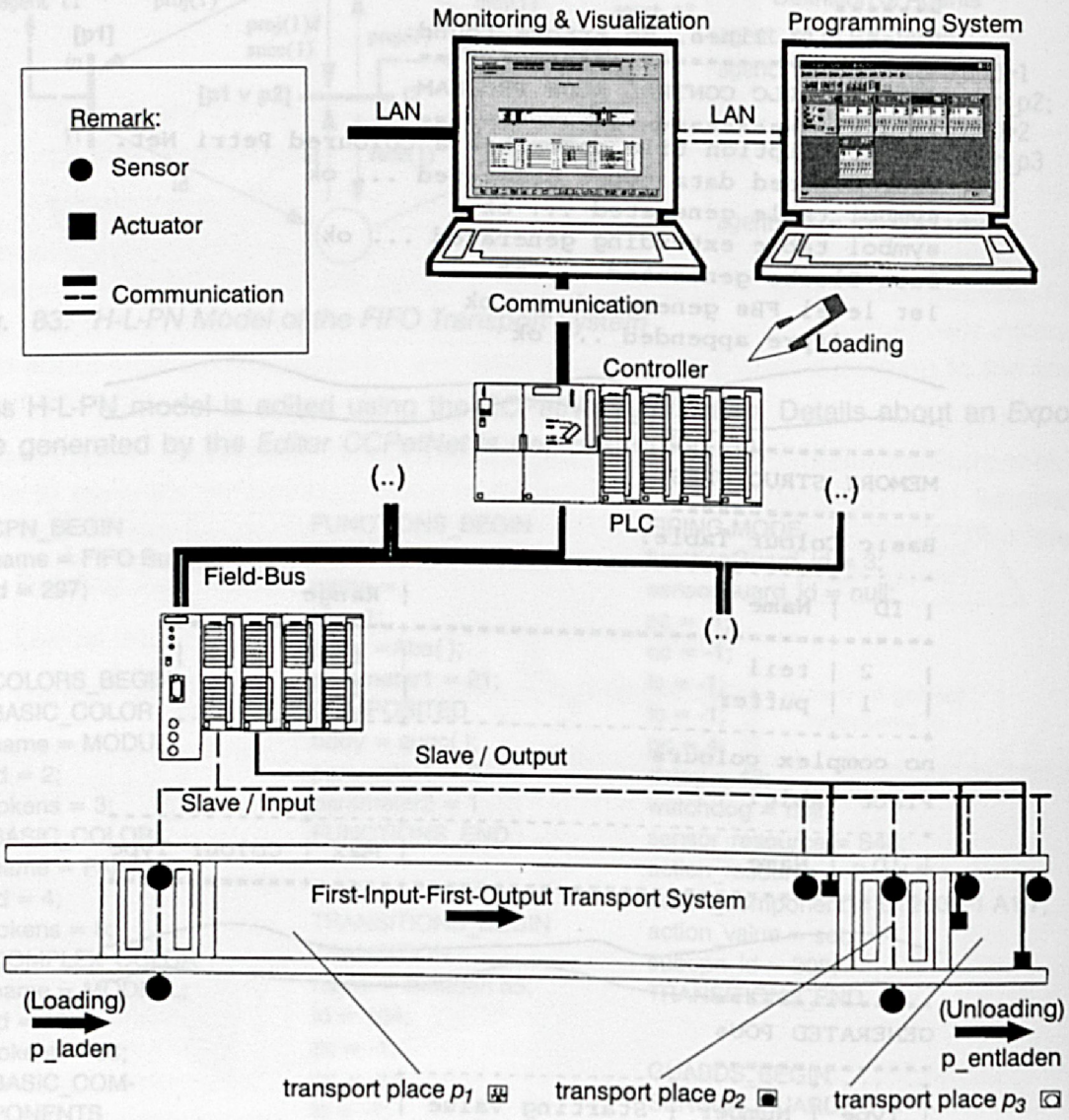


Fig. 86: Implemented Architecture to Control the FIFO Transport System

The hardware-platform-independent characteristics of the developed tool allows the communication between the PLC and other components of the hierarchical control system, e.g., the monitoring and visualization system like the one presented in chapter 6.

7.4 Summary

In the manufacturing industry, programmable logic controllers (PLCs) are a widely used platform for implementing logic control systems. For this reason, the possibility to automatically generate executable PLC-code from the H-L-PN-based logic structures devel-

oped in chapters 3 and 4 was here researched and the results were implemented at industrial level.

By means of the grammar developed here, textual representation of the control logic is derived from the H-L-PN-based description of the logic generated by the editor presented in chapter 6.

The textual description serves then as an input for a compiler, which generates PLC-code according to the standard IEC 1131-Structured Text. Moreover, means for integrating special features of a PLC in the code derived from the H-L-PN were provided. In this context, the H-L-PN grammar provides an interface for calling functions and function blocks defined in one of the languages of the standard IEC 1131-3.

The compiler was integrated in a engineering-tool able to load the generated logic control code into an industrial PLC.

In order to illustrate the applicability of the implemented solution, a well-known component of flexible production systems, i.e., a FIFO transport system, was modelled with H-L-PN and then set into operation and controlled from a PLC loaded with the logic control code, which is generated by the named compiler.

8 Conclusions and Outlook

The factors "time", "flexibility" and "quality" have more and more importance for the existence and success of a company under the hart pressure of international competition. Important advantages can be won only with the shortest development time – from the idea to the final product. The production must meet the demands of the market, but it has to be customers oriented, as well. These tasks can be fulfilled only through the usage of Flexible Production Systems.

The degree of flexibility of these systems depends, not only, on the flexibility of single components, but also, what is even more important, on its control system (DECS). For this reason, the use of complex design methodologies, and control- and monitoring-structures is necessary in order to develop this sort of production systems.

Programmable Logic Controllers (PLC), as well as CNC, NC, RC and PCs, are currently used for running of flexible production systems. It is often to be seen that the usage of different design- and programming-methods for PC-/PLC-solutions results time consuming and therefore costly processes. Design and implementation of control system – tools and methods – is completely separated from the planning, design and implementation of the flexible production system. So, should the FPS layouts or product choice be changed, reprogramming or new programming of the control software is necessary.

The usage of High-Level Petri Nets (H-L-PN) is a new base for design, modeling, validation and implementation of the flexible production systems.

H-L-PN is, opposite to other methods, much more suitable for description of FPS, and discrete, asynchronous and concurrent processes that might occur. H-L-PN has a well-known mathematical theory as base and can graphically show all process events. H-L-PN-based models can be used in the FPS planning-, as well as in the development- and implementation-phase.

The H-L-PN-based model of the FPS is also suitable for the control of production systems. The advantage of this procedure is obvious. Once the H-L-PN model of a FPS is created, a qualitative (structural analysis) and quantitative (e.g., simulation) analysis can be performed in order to validate the system specifications. After this, control logic, of high quality and without development errors, can automatically be generated through derivation of control signals from the validated H-L-PN model.

This work describes an H-L-PN-based approach for FPS development that comprises a new control- and monitoring-concept. Opposite to other solutions, a designed, modelled and validated FPS can be controlled directly through data exchange between H-L-PN-based model and production system, and through information exchange between H-L-PN-based model and overlapped control levels (real-time decisions and/or planning level). The usage of such complete platform-independent and -configurable models of the systems and their control structures makes possible important saves of time and costs.

Two new engineering methods are presented. The first method supports the user in the design and implementation tasks of the H-L-PN- and PC-based control- and monitoring-system of FPS. The second one allows automatic creation of IEC 1131-conform PLC-code out of validated and optimized H-L-PN-based models of FPS. In this case, the resulted control code can be loaded directly in a PLC or an off-line simulator.

The complete FPS/DECS development process, i.e., design, programming and implementation, can be entirely supported through the engineering-tool – based on the H-L-PN theory – developed within this work.

The both above named methods used H-L-PN-based models. For this reason, they are suitable, not only for the development and control of a FPS, but also, because of their graphical and mathematical character, for the visualization of the internal events of the system, and through this for the monitoring/control of the production processes. An extension of the new FPS control concept using an industrial control- and monitoring-component was also developed. This component offers, not only the possibility to show the evolution of H-L-PN-based control model, but allows also the preparation of user-friendly graphic representation of the controlled process. Herewith is achieved the integration of the human operator into the development concept (human-machine system).

Because of their extension abilities and their modular construction, in this work developed and implemented control structures can be used as an excellent base for further ones. Especially, the engineering-tool can be seen as a new FPS component: *a virtual representation of the real flexible production environment*. It is able to run synchronous with the FPS and both can be seen as a unique production entity. Compared with other approaches, the proposed engineering-tool does offer a number of advantages, for instance: specifications can be verified, implementation can be automated, test cases can be generated, labour costs can be saved by reducing working-time of designing a test suite. However, the most important aspect is the possibility to develop and implement more reliable FPS/DECS.

References

/1/ AFCET: "*Normalisation de la representation du cahier des charges d'un automatisme logique*". Final report of AFCET Commission, J. Automatique et Informatique Industrielles. 1977.

/2/ Alla, H.; Ladet, P.; Martinez, J.; Silva, M.: *"Modelling and Validation of Complex Systems by Coloured Petri Nets"*. Adv. in Petri Nets 1984, pp. 15-31. Springer Verlag. 1984.

/3/ Arnold, M.; Buchner, H.; Heim, M.: *"Voraussetzungen und Methoden einer Mensch-Prozeß-Kommunikation"*. Automatisierungstechnik Praxis atp 9. R. Oldenburg Verlag München Wien. 1995.

/4/ Bilinski, K. and Dagles, E.: *"High Level Synthesis of Synchronous parallel Controllers"*. Advances in Petri Nets 1996. Lecture Notes in Computer Science, pp. 93-112. Springer Verlag. 1996.

/5/ Brandin, B.: *"The Real-Time Supervisory Control of an Experimental Manufacturing Cell"*. IEEE Trans. on Rob. and Aut., vol. 12, no. 1, pp. 1-14. 1996.

/6/ Buzacott, J. and Yao, D.: *"Flexible Manufacturing systems: A Review of Analytical Models"*. Management Science, vol. 32, pp. 890-895. 1986.

/7/ Camarinha-Matos, L. and Osorio, L.: *"Monitoring and Error-Recovery in Assembly Tasks"*. Proc. of ISATA, Austria. 1990.

/8/ Camarinha-Matos, L.; Lopes, L.; Barata, J.: *"Integration and Learning in Supervision of Flexible Assembly Systems"*. In IEEE Trans. on Rob. and Aut., vol. 12, no. 2, pp. 202-219. 1996.

/9/ Campos, J.: *"Performance Bounds for Synchronized Queueing Networks"*. PhD Thesis, Dept. de Ing. Eléct. e Informática, Univ. de Zaragoza, Spain. 1990.

/10/ Carelli, R.; Colombo, A. W.; Bernhardt, R.; Schreck, G.: *"Discrete event and motion-oriented simulation for FMS"*. Proc. of the 1st. IEEE/ECLA/IFIP Int. Conf. on Architectures and Design Methods for Balanced Automation Systems BASYS '95. Chapman & Hall. Joint paper: Fraunhofer -IPK- Berlin (Germany) and Instituto de Automatica -INAUT-, San Juan (Argentina). Vitoria, Brasil. 1995.

/11/ Carsten, J.: *"Ein integriertes Steuerungsentwurfs- und Verifikatioskonzept mit Hilfe interpretierter Petri-Netze"*. VDI Reihe 8, num. 641. VDI Verlag. Düsseldorf. 1997.

/12/ Caselli, S.; Papaconstantinou, C.; Doty, K.; Naavthe, S.: *"A structure-function-control paradigm for knowledge-based modeling and design of manufacturing workcells"*. Journal of Intelligent Manufacturing, no. 3, pp. 11-30. Chapman & Hall. 1992.

/13/ Chang, S.; DiCesar, F.; Goldbogen, G.: *"Failure Propagation Trees for Diagnosis in Manufacturing Systems"*. IEEE Trans. on Syst., Man and Cybern., vol. 21, num. 4, pp. 767-775. 1991.

/14/ Chao, D. and Wang, D.: *"Two Theoretical and Practical Aspects of Knitting Technique: Invariants and a New Class of Petri Net"*. IEEE Trans. on Syst., Man and Cybern., vol. 27, num. 6, pp. 962-977. 1997.

/15/ Charwat, J.: *"Lexicon der Mensch-Maschine-Kommunikation"*. 2. verbesserte Auflage. R. Oldenburg Verlag München Wien. 1994.

/16/ Chiola, G.: *"GreatSPN, User's Manual"*. Version 1.3, Dipartamento di Informatica. Universita degli Studi di Torino, Italy. 1987.

/17/ Colombo, A.W.: *"Modelacion y Analisis de Operacion de Sistemas Flexibles de Produccion"* (Master of Science Thesis, in Spanish). In: *Temas de Automatica*, Nr. 2 (Fund. Univ. San Juan, Ed.), San Juan, Argentina. 1995.

/18/ Colombo, A.W.; Martínez, J.; Carelli, R.: *"Formal Validation of Complex Production Systems Using Coloured Petri Nets"*. Proc. of the IEEE International Conference on Robotics and Automation, pp. 1713-1718. San Diego, California, USA. 1994.

/19/ Colombo, A.W.; Martínez, J.; Kuchen, B.: *"Formal Specification and Validation of a Static FIFO Queue Model Using Coloured Petri Nets"*. Proc. of the IEEE International Symposium on Industrial Electronics, pp. 231-235. Santiago, Chile. 1994.

/20/ Colombo, A. W.; Pellicer, J.; Martin, M.; Kuchen, B.: *"Simulador de Sistemas Flexibles de Manufactura usando Redes de Petri"*. Proc. of the Sudamerican Control Conference, pp. 100-102, Rio de Janeiro, Brasil. 1994.

/21/ Colombo, A.W.; Carelli, R.: *"Petri Nets for Designing Manufacturing Systems"*. In Computer-Assisted Management and Control of Manufacturing Systems, cap. 11, pp. 297-324. Ed. S. Tzafestas, Advanced Manufacturing Series, Springer Verlag London. 1997.

/22/ Colombo, A.W.; Carelli, R.; Kuchen, B.: *"A Temporized Petri Net Approach for Design, Modelling and Analysis of Flexible Production Systems"*. Int. J. Adv. Manuf. Technol., vol. 13, no. 3, pp. 214-226. Springer Verlag London. 1997.

/23/ Couvreur, J.M. and Martinez, J.: *"Linear Invariants in Commutative High Level Nets"*. Adv. in Petri Nets 1990. Lecture Notes in Computer Science, vol. 483, pp. 146-165. Springer Verlag. 1990.

/24/ Christiansen, S.; Petrucci, L.: *"Towards a Modular Analysis of Coloured Petri Nets"*. Internal report of Computer Science Dep., Aarhus University, Denmark. 1994.

/25/ David, R. and Alla, H.: *"Petri Nets & Grafcet - Tools for modeling Discrete Event Systems"*. Prentice Hall. 1992.

/26/ David, R.: *"Grafcet: A Powerful Tool for Specification of Logic Controllers"*. IEEE Trans. on Control Systems Technology, vol. 3, num. 3, pp. 253-268. 1995.

/27/ Desrochers, A.: *Modeling and Control of Automated Manufacturing Systems"*. IEEE Computer Society Press. 1990.

/28/ Desrochers, A. and Silva, M.: *"Introduction to the Special Issue on Computer Integrated Manufacturing"*. IEEE Trans. on Rob. and Aut., vol. 10, no. 2, pp. 85-87. 1994.

/29/ DIN EN 60073, IEC 60073, (VDE 0199): *"Grund- und Sicherheitsregeln für die Mensch-Maschine-Schnittstelle, Kennzeichnung"*. Beuth Verlag Berlin. 1997.

/30/ Du, R.; Elbestawi, M.; Wu, S.: *"Automated Monitoring of Manufacturing Processes, Part 1: Monitoring Methods"*. In Journal of Engineering for Industry, vol. 117, no. 121. 1995.

/31/ Duttenhofer, F.; Wegener, U.: *"Echtzeit-Produktionsmanagement - Entscheidungen treffen mit aktueller Fertigungsinformation"*. In Konferenz-Einzelbericht: Workshop über Realzeitsysteme, pp. 120-129. Springer Verlag Berlin, Heidelberg. 1995.

/32/ Ezpeleta, J. and Martínez, J.: *"Petri Nets as a Specification Language for Manufacturing Systems"*. 13th. IMACS World Congress on Computation and Applied Mathematics. Dublin. 1991.

/33/ Ezpeleta, J. and Colom, M.: *"Automatic Synthesis of Colored Petri nets for the Control of FMS"*. IEEE Trans. on Rob. and Aut., vol. 13, num. 3, pp. 327-337. 1997.

/34/ Ehlers, E.M. and Van Rensburg, E.: *"An Object-Oriented Manufacturing Scheduling Approach"*. In IEEE Trans. on Syst., Man and Cybern.-Part A: Systems and Humans, vol. 26, num. 1, pp. 17-26. 1996.

/35/ Fanti, M.; Maione, B.; Piscitelli, G.; Turchiano, B.: *"System Approach to Design Generic Software for Real-Time Control of Flexible Manufacturing Systems"*. IEEE Trans. Syst. Man, Cybern., vol. SMC-A 26, no. 2, pp. 190-202. 1996.

/36/ Feldmann, K.: *"Montageplanung in CIM"*. In CIMMTT: CIM-Fachmann Series, Springer Verlag Berlin. 1992.

/37/ Feldmann, K.: *"Skriptum zur Vorlesung: Rechnerintegrierte Produktionssysteme* (in German)". University of Erlangen, Germany. 1993.

/38/ Feldmann, K.; Colombo, A.W.; Schnur, C.: *"An Approach for Modelling, Analysis and Real-Time Control of Flexible Manufacturing Systems using Petri Nets"*. European Simulation Symposium '95. Erlangen, Germany. 1995.

/39/ Feldmann, K.; Schnur, C.; Colombo, A.W.: *"Modularized, Distributed Real-Time Control of Flexible Production Cells, Using Petri Nets"*. IFAC Journal Control Engineering Practice (CEP)", vol. 4, no. 8, pp. 1067-1078. 1996.

/40/ Feldmann, K.; Colombo, A.W.; Carelli, R.; Kuchen, B.: *"An Approach for Designing Logic Controller Programs using Coloured Petri Net Models"*. Proc. of IFAC Latinamerican Control Conference, vol. 2, pp. 881-886. 1996.

/41/ Feldmann, K.; Schnur, C.; Colombo, A.W.: *"Testing of Logic Control Software for flexible Manufacturing Systems based on 3D-Kinematics-Simulation"*. Proceedings of the 11th. European Simulation Multiconference (ESM), Istanbul, Turkey. 1997.

/42/ Feldmann, K.; Colombo, A.W.; Rauh, E.; Rottbauer, H.: *"Joining Discrete-Event Control and Simulation-based Monitoring for Supervisory Functions"*. Proc. of the IFAC/IFIP Int. Conf. on Management and Control of Production and Logistics (MCPL'97), vol. 1, pp. 205-210. Campinas-SP, Brazil. 1997.

/43/ Ferrarini, L.: *"An Incremental Approach to Logic Controller Design with Petri Nets"*. IEEE Trans. on Syst. Man & Cybernetics, vol. 22, num. 3, pp. 461-473. 1992.

/44/ Freedman, P.: *"Time, Petri nets and Robotics"*. IEEE Trans. on Rob. and Aut., vol. 7, num. 4, pp. 417-433. 1991.

/45/ Gentina, J. and Corbel, D.: *"Coloured Adaptive Structured Petri-Net: A Tool For the Automatic Synthesis of Hierarchical Control of Flexible Manufacturing Systems"*. Proc. of the IEEE Int. Conf. on Robotics and Automation, Raleigh (U.S.A.), pp. 1166-1173. 1987.

/46/ Gershwin, S.; Hildebrant, R.; Suri, R.; Mitter, S.: *"A Control Perspective on Recent Trends in Manufacturing Systems"*. IEEE Control System Magazine, vol. MCS-6, no. 2, pp. 3-15. 1986.

/47/ Giua, A. and DiCesare, F.: *"Petri Net Structural Analysis for Supervisory Control"*. IEEE Trans. on Rob. and Aut., vol. 10, no. 2, pp. 185-195. 1994.

/48/ Grampp, K.: *"Rechnerunterstützung bei Test und Schulung an Steuerungssoftware von SMD-Bestücklinien"*. PhD Thesis, Hanser Verlag München-Wien. 1994.

/49/ Gronau, N.: *"Monitoring in der Production"*. In CIM MANAGEMENT 11-4, pp. 1316. GITO-Verlag. 1995.

/50/ Haddad, S. and Couvreur, J.M.: *"Towards a general and powerful computation of flows for parametrized coloured nets"*. 9th. European Workshop on Applications and Theory of Petri Nets. Venecy, Italy. 1988.

/51/ Hankel, M.: *Programmierung nach IEC 1131-3 Software inclusive Off-line Simulation in PC"*. Bildungspraxis 3. Vogel Verlag. 1995.

/52/ Haugg, F.: *"Software-Engineering und Ihre Qualitätssicherung: Methoden zu erfolgreichen Problemlösungen für den erfolgreichen Personal-Computer-Anwender"*. München:Franzis. 1983.

/53/ Heim, M.: *"Konfigurationsräume in der Mensch-Prozeß-Kommunikation"*. VDI/VDE Jahrbuch'97. VDI Verlag Düsseldorf. 1997.

/54/ Holloway, L. and B. Krogh, B.: *"Synthesis of Feedback Control Logic of a Class of Controlled Petri Nets"*. IEEE Trans. on Aut. Control, vol. 35, no. 5, pp. 514-523. 1990.

/55/ Holloway, L.; Krogh, B.; Giua, A.: *"A Survey of Petri net Methods for Controlled Discrete-Event Systems"*. Discrete-Event Systems: Theory and Applications, vol. 7, no. 2, pp. 151-190. Kluwer Acad. Pub. 1997.

/56/ International Electrotechnic Commission (IEC): *"Draft IEC Publication 1131 - Programmable controllers - Part 5: Communication"*, Pub. 1131-5. 1995.

/57/ Jakoby, W.: *"Automatisierungstechnik - Algorithmen und Programme"*. Entwurf und Programmierung von Automatisierungssystemen. Springer Verlag. 1996.

/58/ Jarschel, W.; Drebinger, A; Bolch, G.: *"Modellierung von Fertigungssystemen mit dem Petri-Netz Simulator PETSY"*. Witschaftsinformatik 34, pp. 535-545. 1992.

/59/ Jensen, K.: *"Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use"*. Vol. 1, Monographs on Theoretical Comp. Sc.. Springer Verlag. 1992.

/60/ Jensen, K.: *"Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use"*. Vol. 2. Springer Verlag. 1995.

/61/ John, K.; Tiegelkamp, M.: "*SPS-Programmierung mit IEC 1131-3*". Springer-Verlag. 1995.

/62/ Kief, H.: *"FFS-Handbuch'92/93. Einführung in Flexible Fertigungssysteme"* (in German). Hanser Verlag München Wien. 1992.

/63/ Kreyszig, E.: *"Introductory Functional Analysis with Applications"*. Ed. Wiley. 1978.

/64/ Kusiak, A.: *"Flexible manufacturing Systems: methods and Studies"*. Elsevier Science Pub.. 1986.

/65/ Langmann, R.: *"Graphische Benutzerschnittstellen - Einführung in die Praxis der Mensch-Prozeß-Kommunikation"*. VDI-Verlag, Düsseldorf. 1994.

/66/ Lauzon, S.; Ma, A.; Mills, J.; Benhabib, B.: *"Application of Discrete-Event-System Theory to Flexible Manufacturing"*. IEEE Control Systems Magazine, vol. 6, num. 1, pp. 41-48. 1996.

/67/ Lee, D. and DiCesare, F.: *"Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search"*. IEEE, Trans. on Rob. and Aut., vol. 10, p.p. 123-132. 1994.

/68/ Lopez, L. and Camarinha-Matos, L.: *"Learning in assembly task execution"*. Proc. of the II European Workshop learning Robots, pp. 129-142. Turin, Italy. 1993.

/69/ Marsam, M.: *"Stochastic Petri Nets: An Elementary Introduction"*. Adv. in Petri Nets 1989, pp. 1-29. Springer Verlag. 1989.

/70/ Marsam, M.; Balbo, G.; Conte, G.; Donatelli, S.; Franceschini, G.: *" Modelling with generalized Stochastic Petri Nets"*. Wiley Series on Parallel Computing. 1995.

/71/ Martínez, J.; Muro, P.; Silva, M.: *"Modeling, Validation and Software Implementation of Production Systems Using High Level Petri Nets"*. Proc. of the IEEE Int. Conf. on Robotics and Automation, Raleigh (U.S.A.), pp. 1180-1185. 1987.

/72/ Memmi, G. and Roucairol, G.: *"Linear algebra in net theory"*. In Lecture Notes in Computer Science, vol. 84, pp. 213-223. Springer verlag N.Y. 1980.

/73/ Moravcik, O. and Misut, M.: *"Decision Support Systems in Manufacturing Systems Management"*. In Computer-Assisted Management and Control of Manufacturing Systems, cap. 2, pp. 57-80. Ed. S. Tzafestas, Advanced Manufacturing Series, Springer Verlag London. 1997.

/74/ Murata, T.: *"Petri Nets: Properties, Analysis and Applications"*. Proceedings of the IEEE, vol. 77, nº 4, p.p. 541-580. 1989.

/75/ Muro, P.: *"Aplication de Tecnicas de Inteligencia Artificial al Diseno de Sistemas Informaticos para el Control de Sistemas de Produccion.* PhD Thesis (in Spanish). Dept. de Ingenieria Electrica e Informatica. Univ. de Zaragoza. Spain. 1990.

/76/ Naylor, A. and Maletz, M.: *"Design of integrated manufacturing system control software"*. IEEE Trans. Syst., Man and Cybern., vol. SMC-17, no. 6, pp. 881-897, 1987.

/77/ N.N.: *INGRES/Windows4GL: Language Reference Manual, Application Editor User's Guide, Programming Guide"*. Alameda. 1992.

/78/ N.N.: *"Concept: Die Programmierung für Industrieautomatisierung"*. AEG-Schneider Automation GmbH. 1997.

/79/ N.N.: *"ACCON-ProSys 1131"*. DELTALOGIC Automatisierungstechnik GmbH. 1997.

/80/ N.N.: *"OpenDK"*. Infoteam Software GmbH. 1997.

/81/ N.N.: *"SIMATIC Automatisierungssysteme: SIMATIC S7 / M7 / C7"*. Katalog ST 70, Siemes AG. 1997.

/82/ N.N.: *"SIMATIC Software: Basissoftware für S7 und M7"*. STEP 7 Benutzerhandbuch, Siemems AG. 1997.

/83/ N.N.: *SIMATIC Software: S7-PLCSIM"*. Programmtest mit simulierter S7-CPU, Siemems AG. 1997.

/84/ N.N.: *"Simulationsbasierte Entscheidungshilfsmittel für Organisation und Produktion"*. Teilprojekt D1 *"Simulation zur Optimierung von Steuerungssoftware für Fertigungssysteme"*. Ergebnisbericht zum 1. Forschungsjahr, Bayerischer Forschungsverbund Simulationstechnik (FORSIM), pp. 183-202. 1996 - 1997.

/85/ N.N.: *http://www.plcopen.org.*

/86/ N.N.: *Design-CPN / Computer Tool for Colored Petri Nets. http://www. daimi.aau.dk/designCPN/*. 1997/98.

/87/ N.N.: *Deneb Robotics. IGRIP User Manual and Tutorials IGRIP 2.4*. 1994.

/88/ Pil, A. and Haruhiko Asada, H.: *"Integrated Structure/Control Design of Mechatronic Systems Using a Recursive Experimental Optimization Method"*. IEEE/ASME Trans. on Mechatronics, vol. 1, num. 3, pp. 191-203. 1996.

/89/ Pimentel, J.: *"Communication Networks for Manufacturing"*. Englewood Cliffs, NJ: Prentice Hall. 1990.

/90/ Pivi, G.; FIAR SpA; Righini, G.: *"FIRST: A Petri net-based software tool for FMS/FAS simulation"*. Proc. of the 5th. CIM Europe Conf., pp. 413-422. Brussels, Belgium. 1989.

/91/ Pritschow, G.; Spur, G.; Weck, M.: *"Leit- und Steuerungstechnik in flexiblen Produktionsanlagen"*. Fortschritte der Fertigung auf Werkzeugmaschinen. Hanser Verlag München Wien. 1991.

/92/ Qiu, G. and Sanjay, J.: *"Structured adaptive supervisory control of a flexible manufacturing system"*. Internal report of the Dept. of Ind. & Manufact. Eng., Pennsylvania State University. USA. 1997.

/93/ Ramswamy, S. and Joshi, S.: *"Distributed Control of Automated Manufacturing Systems"*. In Proc. of the 27th. CIRP International Seminar on Manufacturing Systems. Michigan, USA. 1995.

/94/ Rauh, E.: *"Einbindung der Simulation in das betriebliche Informationsmanagement"*. In Rationelle Nutzung der Simulationstechnik - Entwicklungstrends und Praxisbeispiele, pp. 145-152. H. Utz Verlag Wissenschaft, München. 1997.

/95/ Reithofer, W.; Raczkowsky, J.; Canuto, E.: *"HIMAC-Hierarchical Management and Control in Manufacturing Systems"*. CIMMOD/CIMDEV meeting, Torino, Italy, pp. 1-10. 1994.

/96/ Sahraoui, A.; Atabakche, H.; Courvoisier, M.; Valette, R.: *"Joining Petri Nets and Knowledge based Systems for Monitoring Purposes"*. Proc. of the IEEE Int. Conf. on Robotics and Automation, Raleigh (U.S.A.), pp. 1160-1165. 1987.

/97/ Scheller, J.: *"Modellierung und Einsatz von Softwaresystemen für rechnergeführte Montagezellen"*. PhD Thesis, Hanser Verlag München-Wien. 1991.

/98/ Schnur, C. and Colombo, A.W.: *"Entwicklung einer frei-konfigurierbaren Prozess-Schnittstelle auf Basis von Profibus-DP"*. FAPS-Technologietransfer (TT)-Seminar: Neue Steuerungs- und Sensorkonzepte - Schlüssel zur Leistungssteigerung in der Fertigungsautomatisierung. Universität Erlangen-Nürnberg. Erlangen. 1996.

/99/ Schormüller, M.; Feldmann, K.; Stief, E.: *"Abgesicherte Investition in innovative Fertigungskonzepte mit Hilfe der Simulation"*. In 7. ASIM-Fachtagung "Simulation - Anwendernutzen und Zukunftsaspekte". Verlag Praxiswissen Dortmund. 1996.

/100/ Siemens: *"SIMATIC WinCC: Systembeschreibung, Handbücher"*. 1997.

/101/ Silva, M. : *"Las Redes de Petri en la Automática y la Informática"*. Editorial A.C., Madrid, España. 1985.

/102/ Silva M. and Valette, R.: *"Petri Nets and Flexible Manufacturing"*. Advances in Petri Nets. Lecture Notes in Computer Science, vol. 424, p.p. 374-417. 1989.

/103/ Söte, W.; Neugebauer, B; Kasselmann, S.: *"Visualisierungssysteme im Vergleich"*. Elektronik - Spezial für Ingenieure und Entwickler, num. 21. Franzis-Verlag. 1995.

/104/ Spur, G.; Mertins, K.; Wieneke-Toutaoui; Rabe, M.: *"Modellierung von Informations- und Materialflüssen für die Auslegungsplanung"* (in German). ZwF 85 (1990) 1, pp. 8-13. 1990.

/105/ SPS Magazin: *"Produktübersicht Prozeßvisualisierung"*. Num. 3,, pp. 71-72. 1997.

/106/ Süss, G.: *"Sprachnorm bringt Steuerung in Schwung -Einbindung der internationalen Norm IEC 1131-3 in eine SPS"* (in German). Automatisieren / Steuerungen, pp. 44-48. Published in Elektronik, num. 8. 1993.

/107/ Szafarczyk, M.: *"Monitoring and Automatic Supervision in Manufacturing Systems"*. In Computer-Assisted Management and Control of Manufacturing Systems, cap. 10. Spyros G. Tzafestas (Ed.). Springer Verlag London. 1997.

/108/ Tempelmeier, H.; Kuhn, H.: *"Flexible Fertigungssysteme"*. Springer Verlag. 1992.

/109/ Teruel, E.; Colom, J.; Silva, M.: *"Choice-Free Petri Nets: A Model for Deterministic Concurrent Systems with Bulk Services and Arrivals"*. IEEE Trans. on Syst., Man and Cybern., vol. SMC-A 27, no. 1, pp. 73-83. 1997.

/110/ Tzafestas, S.: *"Modern Manufacturing Systems: An Information Technology Perspective"*. In Computer-Assisted Management and Control of Manufacturing Systems, pp. 1-56. Ed. S. Tzafestas, Advanced Manufacturing Series, Springer Verlag London. 1997.

/111/ Tzafestas, S. and Capkovic, F.: *"Petri Net-Based Approach to Synthesis of Intelligent Control Systems for DEDS"*. In Computer-Assisted Management and Control of Manufacturing Systems, cap. 12, pp. 325-351. Ed. S. Tzafestas, Advanced Manufacturing Series, Springer Verlag London. 1997.

/112/ Valette, R.; Courvoisier, M.; Bigou, J.; Albukerque, J.: *"A Petri net based programmable logic controller"*. IFIP First International Conf. on Computer Application in Production and Engineering, CAPE 83. Amsterdam, Holland. 1983.

/113/ Vasquez, A. and Mirchandani, P.: *"Concurrent resource allocation for production scheduling"*. In Proceedings of the IEEE Int. Conf. on Rob. and Aut., pp. 1060-1066. Sacramento, USA. 1991.

/114/ VDI 94: *"VDI/VDE 3699-5 (Entwurf), Prozeßführung mit Bildschirmen-Meldungen". "VDI/VDE Gesellschafz Meß- und Automatisierungstechnische"*. Düsseldorf. 1994.

/115/ Villarroel, J.: *"Integracion Informatica del Control en Sistemas Flexibles de Fabricacion"*. PhD Thesis, Dept. de Ing. Eléct. e Informática, Univ. de Zaragoza, Spain. 1990.

/116/ Wang F. and Saridis G.: *"Task Translation and Integration Specification in Intelligent Machines"*. IEEE Trans. on Rob. and Aut., vol. 9, p.p. 257-271. 1993.

/117/ Zhou, M. and DiCesare, F: *"Petri Nets synthesis for discrete event control of manufacturing systems"*. Boston, MA: Kluwer Academic Publishers. USA. 1993.

/118/ Zhou, M.: *"Petri Nets in Flexible and Agile Automation"*. New Jersey Institute of Technology. Kluvier Academic Publishers. 1995.

Lebenslauf

Armando Walter Colombo

geboren am 07.08.1960 in Mendoza, Argentinien

verheiratet

1974 – 1973	Grundschule in Mendoza, Argentinien
1974 – 1979	Technische Fachoberschule in Mendoza, Argentinien. Schwerpunktfach: Fertigungstechnik
1980 – 1990	Studium der Elektronik / Elektrotechnik an der Staatlichen Technischen Universität Mendoza, Mendoza, Argentinien. Abschluß 04/1990
1991 – 1994	Postgraduiertenstudium der Regelungstechnik an der Fakultät für Ingenieurwissenschaften der Universität San Juan, Argentinien. Abschluß 11/1994
1992 – 1995	Stipendiat des CONICET (Argentinischer Wissenschaftlicher Rat für Forschung und Technologie) am Instituto de Automatica, Fakultät für Ingenieurwissenschaften der Universität San Juan, Argentinien. Leiter: Prof. Dr.-Ing. B. Kuchen
1995 – 1997	Stipendiat des DAAD (Deutscher Akademischer Austauschdienst) an der Universität Erlangen-Nürnberg
1997 – 1998	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik, Universität Erlangen-Nürnberg. Leiter: Prof. Dr.-Ing. K. Feldmann

ISSN 1431-6226
ISBN 3-87525-109-1