# Modeling and Verification of Reconfigurable Discrete Event Control Systems

**Dissertation**

zur Erlangung des Grades

der Doktorin der Ingenieurwissenschaften

der Naturwissenschaftlich-Technischen Fakultät II
- Physik und Mechatronik -
der Universität des Saarlandes

and

in Partial Fulfillment of the Requirements
for the Joint Ph.D Degree in
Mechatronic Engineering at
XIDIAN UNIVERSITY

by

Jiafeng Zhang

Saarbrücken, Germany / Xi'an, Shaanxi, P. R. China

2015

|  |  |
|---|---|
| Tag des Kolloquiums: | 31.07.2015 |
| Dekan: | Prof. Dr.-Ing. Georg Frey |
| Mitglieder des Prüfungsausschusses: | Prof. Dr. Fernando Tricas García (Vorsitz)<br>Prof. Dr.-Ing. Georg Frey (Gutachter)<br>Prof. Dr.-Ing. Hans-Michael Hanisch (Gutachter)<br>Prof. Dr. rer. nat. Helmut Seidel (weiteres Mitglied)<br>Meng Qin (akademischer Beisitz) |

**Abstract:** Most modern technological systems rely on complicated control technologies, computer technologies, and networked communication technologies. Their dynamic behavior is intricate due to the concurrence and conflict of various signals. Such complex systems are studied as discrete event control systems (DECSs), while the detailed continuous variable processes are abstracted. Dynamic reconfigurable systems are the trend of all future technological systems, such as flight control systems, vehicle electronic systems, and manufacturing systems. In order to meet control requirements continuously, such a dynamic reconfigurable system is able to actively adjust its configuration at runtime by modifying its components, connections among components and data, while changes are detected in the internal/external execution environment. Model based design methodologies attract wide attention since they can detect system defect earlier, increase system reliability, and decrease time and cost on system development. An accurate, compact, and easy formal model to be analyzed is the first step of model based design methods. Formal verification is an expected effective method to completely check if a designed system meets all requirements and to improve the system design scheme. Considering the potential benefits of Timed Net Condition/Event Systems (TNCESs) in modeling and analyzing reconfigurable systems, this dissertation deals with formal modeling and verification of reconfigurable discrete event control systems (RDECSs) based on them.

**Keywords:** Reconfigurable system, Discrete event system, Modeling, Verification, Petri nets.

**Kurzfassung:** Die meisten modernen technologischen Systeme benötigen aufwändige Steuerungs-, Rechner- und Kommunikationstechnologien. Aufgrund von Nebenläufigkeit und Konflikten ergibt sich ein kompliziertes dynamisches Verhalten. Derartige komplexe Systeme werden dadurch untersucht, dass man sie als ereignisdiskrete Steuerungssysteme (Discrete Event Control Systems, DECSs) betrachtet und dabei die detaillierten unterlagerten kontinuierlichen Prozesse abstrahiert. Um die Anforderungen an die Steuerung durchgängig erfüllen zu können adaptieren sich dynamische rekonfigurierbare Systeme zur Laufzeit durch Modifikation ihrer Komponenten, deren Verbindungen untereinander und der gespeicherten Daten, sobald Änderungen in der internen oder externen Umgebung festgestellt werden. Beispiele für dynamische Rekonfigurierbare Systeme finden sich in der Luftfahrt, im Automobilbereich aber auch in Fertigungssystemen. Modellbasierte Entwicklungsmethoden erfreuen sich zunehmender Beliebtheit, da sie es erlauben Fehler früher im Entwicklungs-prozess aufzudecken und damit zu höherer Systemverfügbarkeit bei verkürzter Entwicklungszeit führen. Ein formales Modell des Systems bildet hierbei den ersten wichtigen Schritt. Durch formale Verifikation kann dieses Modell effektiv und vollständig überprüft und ggf. verbessert werden. Eine geeignete Modellform hierfür sind Timed Net Condition/Event Systems (TNCESs). Die vorliegende Dissertation befasst sich mit der Anwendung von TNCES zur Modellierung und Verifikation rekonfigurierbarer ereignisdiskreter Steuerungssysteme (RDECSs).

**Schlüsselworte:** Rekonfigurierbare Systeme, Ereignisdiskrete Systeme, Modellierung, Verifikation, Petrinetze.

**摘要:** 现代技术系统大多依赖于复杂控制技术、计算机技术和网络通信技术。由于信号并发、冲突等的存在，导致系统的动态行为错综复杂。因此，往往将这些系统抽象为离散事件控制系统来研究。动态可重构系统是未来复杂技术系统的趋势，如安全关键的核电系统、飞控系统和普通的汽车电子系统、自动化制造系统等。这种系统可以根据系统内、外执行环境的变化，如部分组件故障、网络阻塞、未知干扰、用户请求等，主动地修改当前构型的部分组件、组件间连接关系和数据等，使当前构型在线地转变为另一种构型，从而保证系统能够持续地满足控制需求。基于模型的设计方法能够较早发现复杂技术系统的设计缺陷、提高系统的可靠性、缩短开发系统所耗费的时间，因此被广泛地关注和应用。一个准确、简洁且易于分析的系统形式化模型是基于模型设计方法的第一步。形式化验证是一种能够全面测试被设计的系统是否满足需求和改进系统设计方案的一种有力措施。考虑到Timed Net Condition/Event Systems (TNCESs)在分析可重构系统方面的诸多潜在优势，本文研究的主要内容就是基于TNCESs的可重构离散事件控制系统的形式化建模与验证。

**关键词：**可重构系统、离散事件系统、建模、验证、Petri网

# ABSTRACT

Most modern technological systems rely on complicated control technologies, computer technologies, and networked communication technologies. Their dynamic behavior is intricate thanks to the concurrence and conflict of various signals. Such complex systems can be studied as discrete event control systems (DECSs) by ignoring their detailed continuous variable processes. Dynamic reconfigurable systems are a trend of all future technological systems, not only limited to safety-critical high-end systems like nuclear control systems and flight control systems, but also other systems like vehicle electronic systems, manufacturing systems, etc. In order to meet control requirements continuously, such a dynamic reconfigurable system is expected to be able to actively adjust its configuration at runtime by modifying its components, connections among components, and data, while changes are detected in the internal/external execution environment. These changes include faults of partial components, blocking on communication networks, unknown disturbances, and user requirements. Model based design methodologies attract wide attention since they can detect system defect earlier during the system design stage, increase system reliability, and decrease time cost on system development. An accurate, compact, and easily to be analyzed formal model is the first step of model based design methods. Based on this, formal verification is an expected effective method to completely check if a designed system meets all requirements and to improve a system design scheme. This dissertation deals with formal modeling and verification of reconfigurable discrete event control systems (RDECSs).

The formalism Net Condition/Event System (NCES) is a modular extension of Petri nets. Its clear modularity fits the modeling requirements of RDECSs. This dissertation studies three possible reconfiguration scenarios of an NCES. They are the modification of places, transitions, and initial markings, respectively. Accordingly, an NCES based nested state machine is developed to cope with their implementation. The correctness of the final NCES model is checked by the software SESA, where the functional properties are specified by Computation Tree Logic (CTL) and its extension extended CTL (eCTL).

It is found that the direct modeling of RDECSs by using NCESs may greatly burden the subsequent verification. To this end, a new formalism Reconfigurable Timed Net Condition/Event System (R-TNCES) is proposed. An R-TNCES is composed of a control module and a behavior module. The behavior module is a set of superposed Timed Net Condition/Event Systems (TNCESs). The control module is a set of reconfiguration functions.

These reconfiguration functions control the switching among TNCESs in the behavior module. Furthermore, considering the similarity among TNCESs of the behavior module, a level-by-level verification method is developed to control the verification complexity of an R-TNCES.

Consistency is one of the most important properties of distributed reconfigurable systems. This dissertation develops a novel coordination method for such systems. All reconfigurable subsystems are modeled by R-TNCESs. A virtual coordinator is built and a communication protocol among it and the subsystems is applied. Such a method has two benefits: 1） an optimal global reconfiguration scenario is obtained and 2) the number of exchanged messages is reduced. All reconfiguration processes are verified with the help of SESA, where functional and temporal properties are specified by CTL, eCTL, and Timed CTL (TCTL).

Finally, in order to analyze the detailed system behavior during dynamic reconfiguration processes of RDECSs. This dissertation further extends the R-TNCES formalism. Reconfiguration functions are newly assigned with action ranges and concurrent decision functions. In addition, the firing rules of events are updated such that the concurrence of reconfiguration events and normal events are conditionally allowed. Similarly, to check the correctness of the extension, SESA is applied. A reconfigurable and energy-efficient vehicle assembly line is applied to illustrate all the work of this part.

**Keywords:** Reconfigurable system, Discrete event system, Modeling, Verification, Petri nets

# 摘要

现代技术系统大多依赖于复杂控制技术、计算机技术和网络通信技术。由于各种信号并发、冲突等的存在，导致系统的动态行为错综复杂。因此，往往将这些系统抽象为离散事件控制系统来研究。动态可重构系统是未来复杂技术系统的趋势，如安全关键的核电系统、飞控系统和普通的汽车电子系统、自动化制造系统等。这种系统可以根据系统内、外执行环境的变化，如部分组件故障、网络阻塞、未知干扰、用户请求等，主动地修改当前构型的部分组件、组件间连接关系和数据等，使当前构型在线地转变为另一种构型，从而保证系统能够持续地满足控制需求。基于模型的设计方法能够较早发现复杂技术系统的设计缺陷、提高系统的可靠性、缩短开发系统所耗费的时间，因此被广泛地关注和应用。一个准确、简洁且易于分析的系统形式化模型是基于模型设计方法的第一步。在此基础上采用形式化验证方法能够全面测试被设计的系统是否满足需求并为改进系统设计方案提供可靠依据。本文研究的主要内容就是可重构离散事件控制系统的形式化建模与验证。

NCES是Petri网的一种模块化扩展形式化体系。其清晰的模块化结构适合于建模可重构离散事件控制系统。本文首先重点研究了三种基于NCES网的可能的重构方案：第一种是对库所的修改，第二种是对变迁的修改，第三种是对系统初始标识的修改。相应地，一种基于NCES的嵌套的状态机被用来分别处理这三种重构方案的控制实施。最终得到的系统模型的正确性由软件SESA检验，其中功能特性由计算树逻辑及其扩展逻辑描述。

考虑到直接用NCES或者TNCES建模可重构控制系统会给后续系统验证带来较大负担, 本文提出了一种新的形式化体系R-TNCES。一个R-TNCES由一个控制模块和一个行为模块组成。其中，行为模块由多个交叠的TNCES组成，控制模块由一组重构函数组成。这些重构函数根据执行环境的改变自动地切换行为模块中的TNCES。此外，考虑到R-TNCES中各个TNCES的相似性，一个分层验证的方法被提了出来，用以控制对R-TNCES验证的复杂性。

为了保证分布式可重构离散事件控制系统的一致性，本文提出了一种新的协调方法。系统中的每一个可重构的分系统都由一个R-TNCES来表示。一个虚拟协调器和该协调器与各分系统间的通信协议被建立起来处理分系统间的一致性问题。这种方法可以达到两个效果：1)给出最优重构方案，2)减少分系统间的通信量。此外，为了保证这种协调方法的正确性，采用了基于计算树逻辑的模型检验技术来验证整个系统的各种行为。

最后，为了有效分析动态重构过程中的系统行为，本文进一步扩展了R-TNCES。

首先，在重构函数中附加了作用范围和并发决策函数。其次，对事件发射规则做了修改，使得重构事件和普通事件能够有条件地并发。同样地，为了保证系统的正确性，本文借助于软件SESA完成对扩展的R-TNCES的形式化验证。一个可重构、节能的汽车装配系统被用来说明这一部分工作的优点和有效性。

**关键词：**可重构系统， 离散事件系统， 建模， 验证， Petri网

# List of Figures

1.1 General structure of a dynamic reconfigurable control system ........... 2

2.1 A composite module of an NCES ........................................ 11
2.2 A screenshot of SESA ................................................... 15

3.1 EnAS demonstrator in Halle [1] ........................................ 20
3.2 Working process of EnAS ............................................... 21
3.3 Policy 1: Production of a tin .......................................... 21
3.4 Policy 2: Production of tins with double pieces ......................... 22
3.5 Specification of the reconfigurable EnAS with NCESs .................. 23
3.6 NCES-based modules for automatic reconfigurations of EnAS ........... 27
3.7 Reachability graph of the reconfigurable architecture .................... 28

4.1 FESTO MPS ................................................................ 34
4.2 Working process of FESTO MPS ......................................... 34
4.3 Control model of $Control_{chain}1$ ......................................... 36
4.4 Allowed reconfigurations of FESTO MPS .................................. 37
4.5 Controller model of a physical process .................................... 38
4.6 Behavior module of $RTN_{FESTO}$ .......................................... 41
4.7 A diagram of a simplified reachability graph of FESTO MPS ........... 45
4.8 Equivalent TNCES model of FESTO's R-TNCES ........................ 48
4.9 Layers of $Light_1$ ........................................................ 50
4.10 The composite module $CM_2^{L1}$ ........................................ 51
4.11 The composite module $CM_4^{L1}$ ........................................ 52
4.12 The composite module $CM_5^{L1}$ ........................................ 52
4.13 The reachability graph generated in the $4^{th}$ step of $Light_1$ .............. 53
4.14 The layers of $Medium$ ................................................... 55
4.15 Improved verification of $Light_1$ ......................................... 58
4.16 Improved verification of 100 steps ....................................... 59

5.1 Architecture of a DRDECS ............................................... 63
5.2 R-TNCES model of $\varepsilon_F$ ................................................ 69

5.3 R-TNCES model of $\varepsilon_E$ .......... 69
5.4 Coordinator of $\varepsilon_F$ and $\varepsilon_E$ .......... 70
5.5 The TNCES model of a distributed reconfiguration process .......... 72

6.1 Default working process diagram of AAS .......... 80
6.2 Possible reconfigurations in AAS .......... 82
6.3 TNCES-based model of $R3$, $R4$, and $W2$ in Mode1 .......... 83
6.4 TNCES-based model of $R3$, $R4$, and $W2$ in Mode4 .......... 84
6.5 State transition graph of Example 1 .......... 84
6.6 State transition graph of Example 2 .......... 85
6.7 A fragment of the reachability graph of Example 3 .......... 89
6.8 Behavior module of $eRN_{AAS}$ .......... 90
6.9 Control module of $eRN_{AAS}$ .......... 92

# List of Tables

4.1 Fundamental structure modification instructions of R-TNCESs ............ 42
4.2 State number in the verification of $Medium$ .............................. 58

5.1 Possible reactions of a subsystem ...................................... 67
5.2 Comparison of exchanged messages ................................... 76

6.1 Time of robots on their energy-efficient modes ........................... 94
6.2 Energy consumption of robots .......................................... 94

# List of Symbols

| | |
|---|---|
| $N$ | An NCES |
| $TN$ | A TNCES |
| $RTN$ | An R-TNCES |
| $p$ | A place of an NCES/a TNCES/an R-TNCES |
| $P$ | A set of places |
| $t$ | A transition of an NCES/a TNCES/an R-TNCES |
| $T$ | A set of transitions |
| $f$ | A flow arc of an NCES/a TNCES/an R-TNCES |
| $F$ | A set of flow arcs |
| $cn$ | A condition signal of an NCES/a TNCES/an R-TNCES |
| $CN$ | A set of condition signals |
| $en$ | An event signal of an NCES/a TNCES/an R-TNCES |
| $EN$ | A set of event signals |
| $V(t)$ | An event-processing mode to $t$ |
| $\boxed{\vee}$ | The event-processing mode $AND$ |
| $\boxed{\wedge}$ | The event-processing mode $OR$ |
| $^{-}t$ | The set of all source places of $t$ |
| $^{\bullet}t$ | The set of pre places of $t$ |
| $t^{\bullet}$ | The set of post places of $t$ |
| $^{\bullet}p$ | The set of pre transitions of $p$ |
| $p^{\bullet}$ | The set of post transitions of $p$ |
| $^{\sim}t$ | The set of all forcing transitions of $t$ |
| $t^{\sim}$ | The set of all forced transitions of $t$ |
| $m$ | A marking of an NCES/a TNCES |
| $d$ | A clock status of a TNCES/an R-TNCES |
| $m(p)$ | Number of tokens in place $p$ |
| $d(p)$ | Clock position of place $p$ |
| $z = (m, d)$ | A state of a TNCES |
| $m_0$ | An initial marking of an NCES |
| $m(P)$ | The sum of tokens of all places in $P$ |
| $R(N, m_0)$ | Reachable set of $N$ from $m_0$ |
| $R(N, z_0)$ | Reachable set of $TN$ from $z_0$ |
| $[R(N, m_0), E]$ | Reachability graph of $N$ from $m_0$ |
| $\mathbb{N}$ | The set of non-negative integers |
| $\mathbb{N}^{+}$ | The set of positive integers |
| $D(z)$ | Delay of state $z$ |
| $[N]$ | The incidence matrix of $N$ |
| $[N]^{+}$ | The output incidence matrix of $N$ |
| $[N]^{-}$ | The input incidence matrix of $N$ |
| $r$ | A reconfiguration function of an R-TNCES |

# List of Abbreviation

| | |
|---|---|
| DEC | Discrete event system |
| DECS | Discrete event control system |
| RDECS | Reconfigurable discrete event control system |
| RCS | Reconfigurable control system |
| DRCS | Dynamic reconfigurable control system |
| NCES | Net condition/event system |
| TNCES | Timed net condition/event system |
| R-TNCES | Reconfigurable timed net condition/event system |
| PLC | Programable logic controller |
| FMS | Flexible manufacturing system |
| CTL | Computation tree logic |
| eCTL | Extended computation tree logic |
| TCTL | Timed computation tree logic |
| CC | Control component |
| DRDECS | Distributed reconfigurable discrete event control system |
| DDECS | Distributed discrete event control system |
| RMS | Reconfigurable manufacturing system |
| REMS | Reconfigurable and energy-efficient manufacturing system |
| AAS | Automatic assembly system |

# Content

ABSTRACT ..... I
摘要 ..... III
List of Figures ..... V
List of Tables ..... VII
List of Symbols ..... IX
List of Abbreviation ..... XI
Chapter 1 Introduction ..... 1
  1.1 Study Object ..... 1
  1.2 State of the Art ..... 5
  1.3 Dissertation Organization ..... 8
Chapter 2 Preliminaries ..... 11
  2.1 Net Condition/Event Systems ..... 11
  2.2 Timed Net Condition/Event Systems ..... 13
  2.3 Model Checking ..... 15
Chapter 3 Possible Reconfigurations in NCESs ..... 19
  3.1 Motivation ..... 19
  3.2 Experimental Manufacturing Platform ..... 20
  3.3 Specification of Reconfigurable Control Systems ..... 22
  3.4 Reconfiguration of Net Condition/Event Systems ..... 25
  3.5 System Verification ..... 27
  3.6 Summary ..... 29
Chapter 4 Reconfigurable Timed Net Condition Event Systems ..... 31
  4.1 Motivation ..... 31
  4.2 Experimental Manufacturing Platform ..... 33
  4.3 Control Components ..... 36
  4.4 R-TNCESs ..... 39
    4.4.1 Definition ..... 40
    4.4.2 Dynamics of R-TNCESs ..... 43
    4.4.3 Reconfiguration Implementation of R-TNCESs ..... 46
  4.5 Verification of R-TNCESs ..... 48
    4.5.1 Verification of the Initial TNCES ..... 49

4.5.2 Verification of Other TNCESs .................................... 53
4.5.3 Verification of the Control Module .............................. 55
4.5.4 System Correctness ............................................ 56
4.5.5 Discussion .................................................... 58
4.6 Summary ......................................................... 59
Chapter 5 Coordination of R-TNCESs ......................................... 61
5.1 Motivation ...................................................... 61
5.2 Reconfigurable Coordination of a DRDECS .............................. 62
5.2.1 Specification of a DRDECS ...................................... 63
5.2.2 Reconfigurable Coordination of a DRDECS ........................ 64
5.3 Modeling of DRDECSs ................................................ 67
5.3.1 Benchmark Production System .................................... 68
5.3.2 Formal Models .................................................. 68
5.4 SESA based Verification of DRDECSs .................................. 72
5.5 Discussion ......................................................... 74
5.6 Summary ............................................................ 76
Chapter 6 Extended R-TNCESs ................................................ 77
6.1 Motivation ......................................................... 77
6.2 Reconfigurable and Energy-efficient Manufacturing Systems ........... 78
6.2.1 System Specification ........................................... 79
6.2.2 Running Example ................................................ 80
6.3 Extended R-TNCESs ................................................... 82
6.3.1 Drawbacks of R-TNCESs .......................................... 82
6.3.2 Extended R-TNCESs .............................................. 86
6.4 Verification of Extended R-TNCESs ................................... 90
6.4.1 Implementation of Extended R-TNCESs ........................... 91
6.4.2 Formal Verification of AAS ..................................... 93
6.5 Summary ............................................................ 95
Chapter 7 Conclusion ....................................................... 97
7.1 Contribution ....................................................... 97
7.2 Discussion and Future Works ......................................... 99
7.2.1 Discussion ..................................................... 99
7.3 Future Work ........................................................ 100
Reference .................................................................. 101

Acknowledgement ........ 109

# Chapter 1 Introduction

Dynamic reconfigurability is an expected property of all future technological systems since it can increase system flexibility and reliability and decrease time cost in developing new products. These technological systems are man-made and rely on complex automatic control technologies. They are always studied as discrete event control systems (DECSs). This thesis reports modeling and verification of reconfigurable DECSs (RDECSs) based on the formalism Net Condition/Event System (NCES). The NCES is a modular extension of well-known Petri nets. As the beginning of a dissertation, this chapter introduces the study object, state of the art on the topic, and the organization of this dissertation.

## 1.1 Study Object

With the trend of globalization, troublous market demands, aggressive competition on a global scale, and rapid changes in process technologies stress enterprises. To remain competitiveness, their manufacturing systems or product systems relying on complex control techniques should meet a new requirement: RECONFIGURABILITY.

A control system with the property of reconfigurability is called a reconfigurable control system (RCS). An RCS is modular and extensible both in its plant and controller. Essentially, it should offer several different but similar configurations in order to meet control requirements in various internal or external conditions. In addition, it should have the ability to change its current configuration due to changed inner/outer environment easily. There are two types of reconfigurations: static and dynamic. The former is always applied off-line before system cold starts. Whereas, the latter is applied automatically at run-time [2]. General aims of a static reconfiguration is to update physical equipments or to integrate new techniques in order to largely improve or modify the original system. However, aims of a dynamic reconfiguration are fault-tolerance or to actively adjust system behavior according to changed environment or user requirements. Generally, a dynamic reconfiguration is a major or minor change to the current configuration but not a change to the system. This dissertation focuses on dynamic reconfigurations.

In fact, the idea of dynamic reconfigurable systems emerged earlier. Historically, from the viewpoint of practical applications, a significant amount of studies on dynamic RCSs were motivated by aircraft flight control systems for the purpose of fault-tolerance [3]. The goal, therein, was to provide self-repairing capability in order to ensure a safe landing in the event of severe faults in the aircraft. Such efforts have been stimulated partially by two commercial aircraft accidents in the late 1970s [4].

A recent study provides another evidence for the need of reconfigurable control. It shows that the fatal crash of EL AL Flight 1862 of a Boeing 747-200F freighter could have been avoided if reconfigurable technologies could be applied. Therefore, a system for aiding pilots by providing automatic fault accommodation is highly desirable for both civil and military aircrafts. In safety-critical nuclear power industries, interests in diagnosis and fault-tolerant control of nuclear power plants have been intensified since the Three Mile Island incident and the tragedy at the Chornobyl nuclear power plant on April 26, 1986. Similar research works had increased progressively since the initial research on reconfigurable (or restructurable) control and self-repairing flight control systems began in the early 1980s [5], [6], [7].

More recently, with the development of communication technologies and computer science, dynamic reconfigurability has begun to draw more and more attention in a wider range of industrial and academic communities, due to increased safety and reliability demands beyond what a conventional control system can offer. Dynamic reconfigurable systems are no longer limited to high-end systems such as aerospace and nuclear power systems. Common products, such as automobiles, are increasingly dependent on microelectronic/mechatronic systems, onboard communication networks, and software, requiring new techniques for achieving dynamic reconfigurability. The objectives of dynamic reconfigurations are not limited to fault tolerance but also to actively adjust system configurations to adapt to frequently changed user requirements or environment.

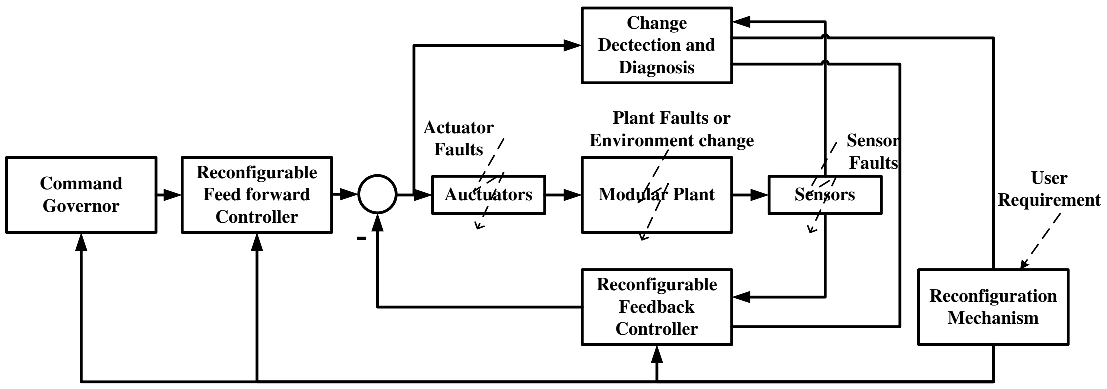

Change Dectection and Diagnosis
Actuator Faults
Plant Faults or Environment change
Sensor Faults
User Requirement
Command Governor
Reconfigurable Feed forward Controller
Auctuators
Modular Plant
Sensors
-
Reconfigurable Feedback Controller
Reconfiguration Mechanism

Fig 1.1 General structure of a dynamic reconfigurable control system

Several review/survey papers on dynamic reconfigurable control systems (DRCSs) have appeared since the 1990s [8], [9], [10]. To summarize, a typical DRCS can be divided into four parts as shown in Fig. 1.1: (1) a modular plant, (2) a reconfigurable controller, (3) a change (disturbances, faults, or user requirements) detection and diagnosis module, and (4) a controller reconfiguration mechanism. In the change detection and diagnosis module, any change such as inside faults and outside environment changes should be detected and isolated as quickly as possible. Furthermore, changed parameters, system state/output variables, and post system models before the detected change need to be estimated on-line in real-time. After that, the reconfigurable controller should be reconfigured automatically to maintain stability, desired dynamic performance and steady-state performance. In addition, in order to ensure the performance of a closed-loop system, a reconfigurable feedforward controller often needs to be synthesized. To avoid potential actuator saturation and to take into consideration the degraded performance after the occurrence of changes, a command/reference governor may also need to be designed to adjust command input or reference trajectory automatically.

The system performance of a DRCS can be evaluated by the transient and the steady-state performance. Here, the transient performance indicates the system performance when the system is in a dynamic reconfiguration process. Whereas, the steady-state performance indicates the system performance when the system is under normal operations, *i.e.*, it is working within a particular configuration. It is important to point out that the emphasis on system behaviors in these two modes of operation can be significantly different. During normal operations, more emphasis should be placed on the system quality of service. In the presence of a fault or a reconfiguration requirement, however, the problem how the system can perform correct behavior during finite time with an acceptable (probably degraded) performance, becomes a predominant issue.

Due to historical reasons and the complexity of the problem, most of the research on fault detection and diagnosis and reconfigurable control was carried out as two separate entities. More specifically, most existing fault diagnosis and identification techniques are developed as diagnostic or monitoring tools, rather than an integral part of a DRCS [4]. This dissertation focuses on the reconfigurable control part. The whole work is carried out by assuming the availability of perfect fault diagnosis and identification techniques and communication techniques among the controller, the plant, and these fault diagnosis and identification modules.

It is common that many modern technological systems can be considered as discrete event systems (DESs). Since 1980s, the development of programable logic controllers (PLCs), computer science and technologies, communication technologies, and sensor technologies has augmented new man-made dynamic systems, mostly technological and often highly complex. Examples around us are air traffic control systems, automated manufacturing systems, computer and communication networks, embedded and networked systems, and software systems. These complicated systems have a common property that they are governed by operational rules designed by humans and their dynamic behavior can be represented as sequences of discrete states if detailed continuous dynamic behavior is ignored. The system states change over time but are driven by discrete events rather than time.

A DES is causal, dynamic, asynchronous, and logical. A DES intentionally ignores some of the system characteristics, specifically, those time-varying detailed continuous processes. However, orders of transient behavior, named events, such as starting or ending a time-varying continuous process are considered in attempting to meet the particular performance specifications. As a result, the evolution of DESs is always described by interactions of discrete events. A discrete event control system (DECS) is a DES, where both the plant and the controller are DESs. The state change of a DECS should follow specified state trajectories or should be always within a specified state area.

It is difficult or improper to describe the dynamic behavior of a DECS by differential or difference equations. Although, some DECSs can be modelled by nonlinear differential or difference equations, the equations are too complex to perform the subsequent system analysis. Therefore, researchers find and propose special tools for DECSs. Typical mathematical analysis tools for DECSs are graph theory [11], temporal logic [12], Petri nets [13], automata [14], as well as some new formalisms [15], [16], [17], [18], [19], [20], [21] based on them. There have been rich studies on DECSs especially after the advent of supervisory control theory of DES that was originated by Ramadge and Wonham in 1987 [22].

A reconfigurable discrete event control system (RDECS) is a DECS. As a reconfigurable system, the controller of an RDECS is required to be reconfigurable. Whether or not the reconfigurable controller can be rapidly reconfigured impacts the straightforward performance of the whole system [23]. An RDECS has several different but similar configurations. Each of these configurations is a DECS and has its own state-space. The reconfigurable controller will decide which configuration can be activated to meet control requirements in changed external/internal conditions. The most important problem is that, the reconfigurable controller should also deal with the dynamic reconfiguration processes, *i.e.*, the switching processes from one configuration to another configuration, to guarantee the correctness and safety of the whole system.

Developing a new RDECS typically includes the following tasks: requirements, modeling, control design, code generation, implementation, hardware-in-the-loop simulation, commissioning, operation, and reconfiguration. Validation, verification and testing are inserted between the different tasks since it is expensive to find errors late in the design process. Obviously, models are key elements of the procedure. The advantage of using models is that fewer prototypes have to be built. This dissertation focuses on modeling and verification of RCSs by considering them as RDECSs.

## 1.2 State of the Art

In order to perform accurate analysis and to improve the system performance of an RDECS, a proper mathematical model is the first critical step. Petri nets [24] are a popular and widely used mathematical tool for handling fault detection and control problems in DECSs. For example, in flexible manufacturing systems (FMSs), Petri net techniques are used to cope with deadlock control problems [25], [26], [27], [28] and fault diagnosis [29], [30].

The formalism Net Condition/Event System (NCES) [21] is an extension of Petri nets, which was first introduced by Rausch and Hanisch in 1995 [31] for modeling of real-world industry control systems. NCESs have been greatly developed in recent years, especially in the research work about IEC61499 as well as the corresponding applications [32], [33]. NCESs support the way of thinking and modeling a system as a set of modules with particular dynamic behavior and their interconnections via signals. The extra condition/event signals and the non-interleaving semantics, *i.e.*, the possibility of firing several transitions simultaneously, make it more powerful than Petri nets in modeling distributed processes and their interactions [34]. The NCES formalism is enriched to Timed Net Condition/Event System (TNCES) by assigning a time interval to each output flow arc. Specially, each place bears a clock which is running if the place is marked. All running clocks run at the same speed measuring the time while the token state of its place remains unchanged. The self-modularity of the TNCES formalism potentially coincides with the modularity of RDECSs. However, the same as Petri nets, the TNCES formalism is for systems that cannot be reconfigured and the behavior such as the modification of places, transitions, arcs, condition/event signals, and initial markings cannot be described directly.

Recently, many researchers have tried to deal with the modeling of control systems with potential reconfigurations. They focus primarily on two directions: direct and indirect. Direct methods offer reconfiguration mechanisms or specific rules coping with system structure modifications, whereas indirect methods usually import extra mechanisms to describe system reconfigurations.

Valk develops self-modifying nets [35] that can modify their own firing rules such that the computational power of ordinary Petri nets is increased. Guan and Lim develop reconfigurable Petri nets (RPN) [36] as a modeling formalism for auto-modified multimedia and execution protocols, where a special place called a modifier was proposed to deal with the reconfiguration behavior. Llorens and Oliver propose net rewriting systems [37] that extend the basic model of Petri nets, making possible the description of dynamic changes in concurrent systems. Their work is improved in [38] by classifying net blocks according to their interfaces in order to guarantee the correctness of a reconfigurable Petri net such as boundedness and liveness. Almeida *et al.* [39] develop an event-condition-action (ECA) paradigm for the design of reconfigurable logic controllers. Their study shows that the reconfigurability is highly dependent on the level of modularity of the logic control system, and that not all "modular" structures are reconfigurable. Wu and Zhou [40] present intelligent token Petri nets (ITPN). In their model, tokens representing job instances carry real-time knowledge about system states and changes just like smart cards in practice such that dynamical changes of a system can be easily modeled. All these formalisms can describe the system's reconfiguration behavior. However, some of them do not clearly define the modularity which brings complexity in designing, understanding, and future redevelopment. The system's correctness such as coherence of states before and after system reconfigurations is not considered. Moreover, temporal constraints that are of great importance are not mentioned.

Sampath *et al.* [41] define a reconfiguration method for a class of discrete event systems (DESs) that are subject to linear constraints as their control specifications. This method is suitable for non real-time reconfigurable systems such as hospital management systems. Dumitrache *et al.* [42] propose a real-time reconfigurable supervised control architecture for large-scale manufacturing systems in order to evaluate and improve the performance of the control architecture. Ohashi and Shin [43] develop a model-based control design for reconfigurable manufacturing systems by using state transition diagrams and a general graph representation, taking the reconfiguration and reuse of design data into account. Kalita and Khargonekar [44] define a hierarchical structure and a framework for modeling, specification, analysis, and design of logic controllers for reconfigurable manufacturing systems (RMSs), which allows reusability and rapid reconfigurability of the controller while the machining system is reconfigured. Liu and Darabi [45] develop a discrete event controller based on finite automata, which can be reconfigured by a mechanism called mega-controller, as responses to local sensor failures. In the literature, some other methods also offer the descriptions of complex dynamic systems, such as the mobile nets [46] and holonic systems [47], [48]. These studies have tried to describe the reconfigurability and to reflect the characteristics of RDECSs. Nevertheless, some of them do not consider the temporal constraints. Most of them cannot ensure the correctness or validity especially the coherence of states in an RDECS. None of them deal with the reconfigurations based on the TNCES formalism. Furthermore, most of the indirect methods cannot represent an RDECS in a compact manner.

The consistency during a reconfiguration is one of the most important required properties of an RDECS. Several research works have been done in recent years, which define inter-agent communication protocols for the coordination of various components in an RCS [49], [50], [51]. The authors of [52] develop KB-ORG that is a fully automated, knowledge-based organization designer for multi-agent systems. It uses both application-level and coordination-level organization design knowledge to explore the combinatorial search space of candidate organizations selectively. This approach significantly reduces the exploration effort required to produce effective designs. The studies of [53] and [54] propose a cooperative mediation based negotiation protocol, called asynchronous partial overlay (APO). It allows the agents to extend and overlap the context that they use for making their local decisions. This variable based decomposition technique allows for rapid distributed asynchronous problem solving without the explosive communications overhead normally associated with this decomposition technique. The work in [55] develops a collaborative network for atmospheric sensing (CNAS), which is an agent-based, power-aware sensor network for ground-level atmospheric monitoring. CNAS agents must have their radios turned off most of the time, as even listening consumes significant power. All these communication modes are effective in their application fields. However, the rate of exchanged messages is an important criterion in order to guarantee an acceptable level of satisfaction and robustness [50], [51]. A multi-agent architecture is proposed in [56] to deal with the coherence of distributed devices, where the exchanged messages among agents are reduced remarkably compared with a direct point-to-point communication mode among agents. The method in [56] gives a coordination solution for multiple concurrent requirements. However, the solution only aims to satisfy the requirement with the highest priority. An optimal coordination solution for all concurrent reconfiguration requirements is not studied.

Verification is another critical step in model based design of complex systems. Its aim is to perform the act of reviewing, inspecting or testing, in order to establish and document that a product, service or system meets regulatory or technical standards. An RDECS may generate a huge state space no matter how simple it looks. It is necessary to perform formal verification on it, the results of which will help designers to improve the designing scheme. Finite state machines are widely used for the modeling of control flow in embedded systems and are amenable to formal analysis like model checking [57], [58]. Model checking is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas before efficient symbolic algorithms are used to traverse the model defined by the system and to check whether the specification holds or not. Extremely large state-spaces can often be traversed in minutes. The technique has been applied to several complex industrial systems such as the Future bus and the PCI local bus protocols. Two kinds of computational tools have been developed last years for model checking: tools like KRONOS [59], UPPAAL [60], HyTech [61], and SESA [62], [63], which compute sets of reachable states exactly and effectively, whereas emerging tools like CHECKMATE [64], d/dt [65], and level-sets methods [66] approximate sets of reachable states. Several research works have been proposed in recent years to control the verification complexity by applying hierarchical model checking for complex embedded systems [32], [67]. Nevertheless, not much attention is paid to TNCESs or reconfiguration forms such as the modification of condition/event signals that can be applied at run-time.

## 1.3 Dissertation Organization

The automatic control technologies used in these complex RDECSs play a rather important role for their availability and reliability. In order to reduce the startup time to a new product and to increase the reliability and safety of a complex technological system, a model-based design methodology is always applied, where to obtain an accurate formal model is the first critical step. On this basis, the formal verification is carried out for testing and improving a design scheme. Considering the lacking of a proper formalism for modeling RDECSs and the potential advantage of NCESs/TNCESs in modeling RDECSs, this particular research copes with formal modeling and verification of RDECSs based on NCESs/TNCESs. The dissertation is organized as follows.

Chapter 2 recalls basic definitions and properties of NCESs and TNCESs. The fundamentals of model checking technologies are briefly introduced. Especially, the modal logic computation tree logic (CTL), extended CTL (eCTL), and timed CTL (TCTL) as well as their instruction in the software SESA are represented.

Chapter 3 describes possible reconfiguration scenarios and their control implementation in an industrial control system that is modeled by an NCES. Here, the reconfiguration scenarios include modification of places, modification of transitions, and updating of initial states. Their control implementation is realized by an NCES based nested state machine.

Chapter 4 represents the formalism Reconfigurable TNCES (R-TNCES) for modeling and verification of RDECSs. It is a reconfigurable extension of the formalism TNCES. In addition, an optimal verification method is shown in this chapter for a special class of R-TNCESs.

Chapter 5 depicts a coordination method for a distributed reconfigurable discrete event control system (DRDECS), where each reconfigurable subsystem is modeled by an R-TNCES. The coordination is implemented by a virtual coordinator together with a communication protocol between it and the subsystems.

Chapter 6 reports the extended R-TNCES formalism. Considering some disadvantages of R-TNCESs in modeling and analyzing dynamic reconfiguration processes of an RDECS, reconfiguration functions in an R-TNCES are newly assigned with action ranges and concurrent decision functions. The new extension is applied to a reconfigurable vehicle assembly line for the aim of energy-saving.

Finally, Chapter 7 concludes this dissertation. Specially, the advantages and identified shortcomings of the current work are discussed. Moreover, prospective future work based on the findings of this dissertation is introduced briefly.

# Chapter 2 Preliminaries

This dissertation reports formal modeling and verification of reconfigurable discrete event control systems (RDECSs). All independent innovation works relating to modeling are based on the formalisms Net Condition/Event System (NCES) and Timed Net Condition/Event System (TNCES). Model checking technologies are applied to perform the formal verification. For a better understanding of works of this dissertation, relevant elemental knowledge on NCESs, TNCESs and model checking technologies are recalled in this chapter.

## 2.1 Net Condition/Event Systems

The formalism Net Condition/Event System (NCES) is an extension of Petri nets. It was introduced by Rausch and Hanisch in [31] and further developed in [21]. An NCES is characterized by clear modular structures. A basic module of an NCES is a typical Petri net [24], *i.e.*, it is composed of places, transitions, flow arcs, and tokens. Each basic module of an NCES interconnects with other modules via special condition/event signals, which make an NCES different from a Petri net. Multiple interconnected basic modules can form a composite module (as seen in Fig. 2.1). A composite module without any input/output condition/event signal is called an autonomous NCES. An autonomous NCES is a place-transition net formally represented by a 7-tuple:

$$\mathrm{N} = (P, T, F, CN, EN, V, m_0),$$

where,

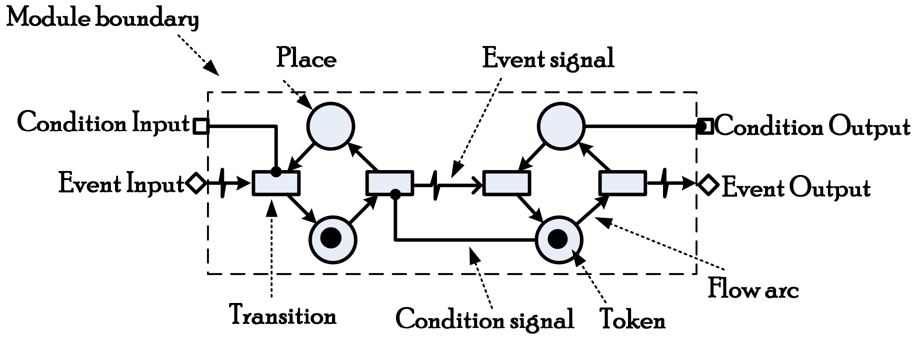


Fig 2.1 A composite module of an NCES

- $P$ is a non-empty finite set of places.
- $T$ is a non-empty finite set of transitions.
- $F$ is a set of flow arcs, $F \subseteq (P \times T) \cup (T \times P)$.
- $CN$ is a set of condition signals with $CN \subseteq (P \times T)$ (*resp*, $EN \subseteq (T \times T)$).
- $EN$ is a subset of $(T \times T) \setminus id_T$, the irreflexive signal (flow) relation.
- $V : T \to \{\boxed{\triangledown},\boxed{\triangle}\}$ maps an event-processing mode (AND or OR) to each transition.
- $m_0 : P \to \{0, 1\}$ is the initial marking.

Generally, $m(p)$ denotes the token state of place $p$. The sum of tokens of all places in $P$ is denoted by $m(P)$, *i.e.*, $m(P) = \sum_{p \in P} m(p)$. Let $x \in P \cup T$ be a node of an NCES. The preset (*resp*, postset) of $x$ is defined as ${}^{\bullet}x = \{y \in P \cup T | (y, x) \in F\}$ (*resp*, $x^{\bullet} = \{y \in P \cup T | (x, y) \in F\}$). A place $p$ is called a source place of a transition $t$ if there is a condition signal from $p$ to $t$. ${}^{-}t$ denotes the set of source places of $t$. Transitions in an NCES are classified into two types: spontaneous and forced transitions. A transition $t' \in T$ is called a forcing (*resp*, forced) transition of transition $t$ if there is a event signal from $t'$ to $t$ (*resp*, from $t$ to $t'$). ${}^{\sim}t$ (*resp*, $t^{\sim}$) denotes the set of forcing transitions (*resp*, forced transitions) of $t$. The set of spontaneous transitions is denoted by $St$, whereas the set of forced transitions is denoted by $Ft$. It is defined that $T = Ft \cup St$ and $Ft \cap St = \emptyset$.

The semantics of NCESs are defined by the firing rules of transitions. A transition $t$ is said to have a token concession if $\forall p \in {}^{\bullet}t$, $m(p) = 1$. It is said to have a condition concession if $\forall p \in {}^{-}t$, $m(p) = 1$. A transition is enabled if it has both token concession and condition concession. A spontaneous transition can fire if it is enabled. However, for an enabled forced transition $t$, in the case of $V(t) = \boxed{\triangle}$, $t$ can fire only after all transitions in ${}^{\sim}t$ fire. In the case of $V(t) = \boxed{\triangledown}$, $t$ can fire only after at least one of the transitions in ${}^{\sim}t$ fires.

In addition, an NCES is executed in steps, *i.e.*, sets of transitions are fired simultaneously. Executable steps are formed by first picking up a nonempty set of enabled spontaneous transitions and then adding as many as possible of those transitions that are forced to fire by signal-events produced by transitions in the step. This implies that in every non-dead NCES, there exists a spontaneous transition. To make this more precise the signal-completeness of transition sets is defined inductively:

Basis: Every subset $s \subseteq St$ is signal-complete.

Step: If $s \subseteq T$ is signal-complete, $t \in Ft$, $V(t) = \boxed{\triangledown}$ and $St \cap s \neq \emptyset$ OR $V(t) = \boxed{\triangle}$ and $St \subseteq s$, then $s \cup t$ is signal-complete.

Obviously, the empty set is signal-complete and $\emptyset$ is the only signal-complete set containing no spontaneous transition. A signal-complete set of transitions may fire simultaneously as far as signal-events are concerned.

A transition $t \in Ft$ is said to be forced by the set $s$ iff $t \notin s$ and $s \cup \{t\}$ is signal-complete.

• A subset $s \subseteq T$ is said to be a step of $N$ iff

1. $s \cap St \neq \emptyset$ ($s$ is signal-founded, *i.e.*, there is at least one spontaneous transition in $s$), and

2. $s$ is signal-complete (*i.e.*, all necessary signal-events will occur).

• A step $s$ of $N$ is called enabled at the marking $m$ iff

3. $^{\bullet}s \leq m$ ($s$ has token-concession, *i.e.*, the transitions in $s$ are concurrently enabled w.r.t. tokens), and

4. $^{-}s \leq m$ (*i.e.*, the conditions of all $t \in s$ are satisfied)

• A step $s$ of $N$ is said to be executable at the marking $m$ iff $s$ is enabled at $m$ and

5. there is no forced transition $t \in Ft$ such that $s \cup \{t\}$ also satisfies 1-4 ($s$ is signal-closed, *i.e.*, maximal with respect to inclusion of forced transitions).

A forced transition $t$ with $V(t) = \boxed{\triangle}$ appears in an enabled step only if it receives signals from all its signal sources. Otherwise, a forced transition t with $V(t) = \boxed{\triangledown}$ appears in an enabled step if it receives a signal from at least one of its signal sources.

If $s$ is an executable step at $m$, then $s$ may fire, which leads to a new state of $N$, *i.e.*, the marking $m' := m - s^{\bullet}+^{\bullet}s$. This is abbreviated as $m \xrightarrow{s} m'$. The reachability relation is defined as usual. Let $R(N, m_0)$ denote the set of all reachable markings of the NCES $N$ from $m_0$. The reachability graph is a structure $[R(N, m_0), E]$ where $E$ is the set of edges such that $(m, m') \in E$ iff $m, m' \in RN(m_0)$ and there is a step $s$ with $m \xrightarrow{s} m'$.

## 2.2 Timed Net Condition/Event Systems

The NCES formalism is enriched in the past years to consider time constraints that are applied to input arcs of transitions: to every pre-arc of a transition, an interval $[eft, lft]$ of natural numbers is attached with $0 \leq eft < lft \leq w$ ($w$ is a given integer). The interpretation is as follows: Every place $p$ bears a clock that is running (*resp*, switched) if the place is marked (*resp*, unmarked). All running clocks run at the same speed measuring the time of the token states (*i.e.*, the clock on a marked place $p$ shows the age of the youngest token in $p$). If firing transition $t$ removes a token from place $p$ or adds a token to $p$, the clock of $p$ is turned back to 0. In addition, a transition $t$ is able to remove tokens from its pre-places (*i.e.*, to fire) only if $\forall p \in {}^{\bullet}t$, the clock at place $p$ shows a time $D(p)$ such that $eft(p,t) \leq D(p) \leq lft(p,t)$. Hence, the firing of transitions is restricted by the clock positions. This extended formalism is called Timed NCES (TNCES).

In this dissertation, $\mathbb{N} = \{0, 1, 2, ...\}$ denotes the set of nonnegative integer and $\mathbb{N}^+ = \{1, 2, ...\}$ denotes the set of positive integer. A state of a TNCES is denoted by $z = (m, d)$, where $m : P \to \{0, 1\}$ (*resp*, $d : P \to \mathbb{N}$) is the token states (*resp*, clock positions) of places. We usually describe markings and time positions using a multiset (bag) or formal sum notation for economy of space. As a result, $\sum_{p\in P} M(p)p$ (*resp*, $\sum_{p\in P} D(p)p$) is used to denote vector $M$ (*resp*, $D$).

The firing rules of TNCESs are a combination of the ordinary firing rules of Petri nets and firing of maximal steps [31], where a maximal step is a set of transitions that can be fired simultaneously. The set of states reachable from $z_0$ is called the reachability set of a TNCES $TN$, which is denoted by $R(TN, z_0)$.

A TNCES is executed in steps too. The execution of a step does not take time. Let $(m, d)$ be a state. A step $s$ of $TN$ is said to be enabled at the state $(m, d)$ of $TN$ (compare this to Section 2.1) iff

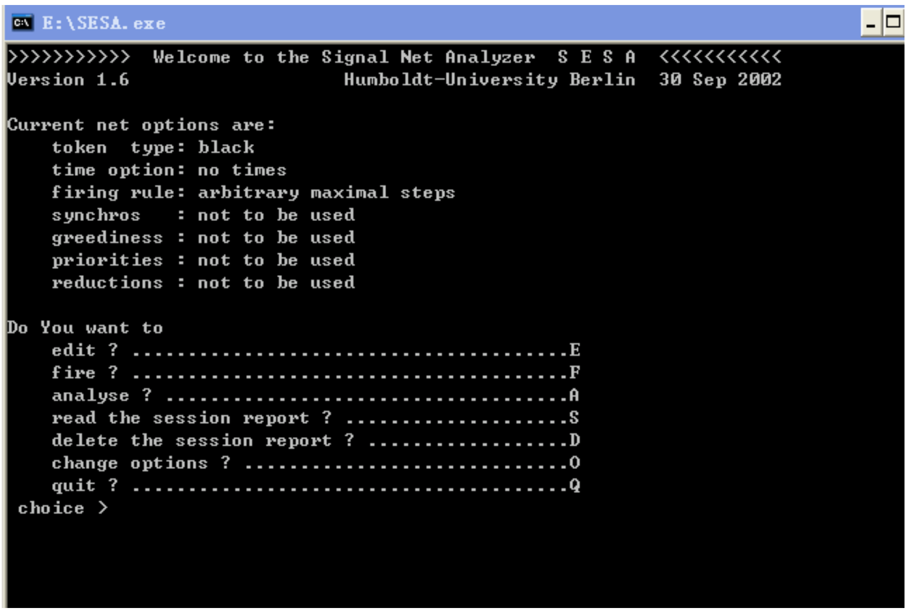
1. ${}^{\bullet}s \leq m$ and for every pre-place $p$ of a transition $t \in s$ it holds $eft(p,t) \leq d(p) \leq lft(p,t)$ (*i.e.*, $s$ has token-concession and the clocks are between $eft$ and $lft$), and
2. ${}^{-}s \leq m$.

Obviously, a step $s$ may be enabled at the marking $m$ in $N$, but not enabled at the state $z = (m, d)$ of $TN$ because some clocks have not reached the earliest firing time $eft$ or have passed already the latest firing time $lft$. The state $z = (m, d)$ of a TNCES may change not only by execution of a step but also by elapsing of one time unit to $z' = (m', d')$, where

$$D'(p) := \begin{cases} D(p) + 1 & m'(p) = m(p)\ \&\ m(p) \neq 0 \\ 0 & \text{else} \end{cases} \tag{2-1}$$

If at a state $z = (m, d)$ of $TN$, no step is enabled or can become enabled by elapsing of time then this state is called *dead*. Otherwise, the minimal number of time units after which at least one step becomes enabled is called the delay $D(z)$ of the state $z = (m, d)$. Hence, the delay is defined only for non-dead states.

Since every executable step has to contain a spontaneous transition, the delay of a non-dead state is the minimal number of time units after which at least one spontaneous transition becomes enabled. This number obviously may be zero.



```
E:\SESA.exe
>>>>>>>>>>> Welcome to the Signal Net Analyzer  S E S A  <<<<<<<<<<<
Version 1.6                    Humboldt-University Berlin  30 Sep 2002

Current net options are:
    token  type: black
    time option: no times
    firing rule: arbitrary maximal steps
    synchros    : not to be used
    greediness  : not to be used
    priorities  : not to be used
    reductions  : not to be used

Do You want to
    edit ? ......................................E
    fire ? ......................................F
    analyse ? ...................................A
    read the session report ? ...................S
    delete the session report ? .................D
    change options ? ............................O
    quit ? ......................................Q
 choice >
```

Fig 2.2 A screenshot of SESA

Let $z = (m, d)$ be a non-dead state. Following the weak earliest firing rule we call a step $s$ to be executable at state $z$ iff $s$ is enabled after elapsing of $D(z)$ time units. Given a non-dead state $z = (m, d)$ we first compute the delay $D(z)$ and elapse $D(z)$ time units resulting in the state $z' = (m', d')$. Next the set $E$ of all spontaneous transitions enabled at $z'$ is computed. Then we proceed with $E$ like the normal firing rule does, resulting in a list of executable steps. These steps are considered as executable at the original state $z$ (they all have the delay $D(z)$). The execution of an executable step $s$ at the state $z$ then is done by first elapsing $D(z)$ time units and then firing $s$. The state $z'' = (m'', d')$ reached by the execution of $s$ is determined by $m'' = m - s^{\bullet} + {}^{\bullet}s$, and

$$d'(p) := \begin{cases} d(p) + D(z) & \text{if } m(p) > 0 \wedge m'(p) > 0 \wedge \text{p} \notin (\text{Fs} \cup \text{sF}) \\ 0 & \text{else} \end{cases} \tag{2-2}$$

## 2.3 Model Checking

Model checking [57], [58] is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check whether the specification holds or not. Extremely large state-spaces can often be traversed in minutes. The technique has been applied to several complex industrial systems such as the Future bus and the PCI (Peripheral Component Interconnect) local bus protocols.

SESA [62] is an effective tool for the analysis of NCESs and TNCESs. A screenshot of it is show in Fig. 2.2. Typical properties which can be verified are boundedness of places, liveness of transitions, and reachability of markings or states. General properties can be expressed in CTL and verified by the model checker of SESA. To reduce the size of the state space and the time for its construction, SESA offers several reduction methods. SESA also derives some analysis results from the underlying Petri net of a signal-net system. SESA inherited much of its code from the Petri net tool INA [68].

In this dissertation, SESA [62], [63] is applied to do the verification of all addressed issues, where Computation Tree Logic (CTL) [69] and extended Computation Tree Logic (eCTL) [70] are used to specify functional properties of an RDECS, and Timed Computation Tree Logic (TCTL) [67] is used to describe temporal properties. In CTL, all formulae specify the behavior of a system starting from an assigned state, in which formulae are evaluated by taking paths, (*i.e.*, sequences of states) into account. A formula holds for the system if it is evaluated to be true in the initial state of the system. The model-checker SESA is a rich tool to analyze and verify functional and temporal properties of NCESs and TNCESs. This section briefly introduces the syntax and semantics of CTL, eCTL, TCTL, and their state predicates as well as atomic state propositions used in SESA.

The semantics of CTL formulae are defined with respect to a reachability graph, where states and paths are used for the evaluation. A reachability graph $G$ consists of all global states that the system can reach from a given initial state. It is formally defined as a tuple $G = [Z; E]$, where $Z$ is a finite set of states, and $E$ is a finite set of transitions between states, *i.e.*, a set of edges $(z; z')$ such that $z$, $z' \in Z$ and $z'$ is reachable from $z$.

In CTL, paths play a key role in the definition and evaluation of formulae. A path denoted by $(z_i)$ starting from the state $z_0$ is a sequence of states, $(z_i) = z_0 z_1 ...$ such that $\forall j \in \mathbb{N}$, there is an edge $(z_j; z_{j+1}) \in E$. The truth value of a CTL formula is evaluated with respect to a certain state of the reachability graph. Let $z_0 \in Z$ be a state of the reachability graph and $\varphi$ be a CTL formula. The relation $z_0 \models \varphi$ means that the CTL formula $\varphi$ is satisfied in the state $z_0$. Then the relation $\models$ for a CTL formula is defined as follows:

- $z_0 \models EF\varphi$, if there is a path $(z_i)$ and $j > 0$ such that $z_j \models \varphi$,
- $z_0 \models AF\varphi$, if for all paths $(z_i)$, there exists $j > 0$ such that $z_j \models \varphi$,
- $z_0 \models AG\varphi$, if for all paths $(z_i)$ and for all $j > 0$, it holds $z_j \models \varphi$.

In CTL, it is rather complicated to refer to the information contained in certain transitions among states of a reachability graph. A solution is given in [70] by proposing an extension to CTL called eCTL. A transition formula is imported in eCTL to show the transition information contained in the edges of the reachability graph. Therefore, the structure of the reachability graph $G = [Z; E]$ is improved, where $Z$ is a finite set of states, and $E$ is a finite set of transitions among states, *i.e.*, a set of labeled edges $(z, u, z')$, such that $z$, $z' \in Z$ and $z'$ is reachable from $z$ by executing the set of transitions $u$.

Let $z_0 \in G$ be a state of the reachability graph, $\psi$ be a transition formula, and $\varphi$ be an eCTL formula. Then the relation $\models$ for eCTL formulae are defined inductively:

- $z_0 \models E\psi X\varphi$, if there exists a successor state $z_1$ such that there is an edge $(z_0, u, z_1) \in E$, where $(z_0, u, z_1) \models \psi$ and $z_1 \models \varphi$ holds,
- $z_0 \models A\psi X\varphi$, if $z_1 \models \varphi$ holds for all successors states $z_1$ with an edge $(z_0, u, z_1) \in E$ such that $(z_0, u, z_1) \models \psi$ holds.

TCTL is an extension to CTL to model qualitative temporal assertions together with time constraints [67]. For a reachability graph $G = [Z; E]$, the state delay $Delay$ is defined as a mapping $Delay : Z \to \mathbb{N}$. $\forall z$, $Delay(z)$ denotes the number of time units that have to elapse at $z$ before firing a transition from this state. For a path $(z_i) = z_0z_1, ...$ and a state $z \in Z$, we have

- $Delay[(z_i), z] = 0$ if $z_0 = z$,
- $Delay[(z_i), z] = Delay(z_0)+Delay(z_1)+...+Delay(z_k)$ if $z_k = z$ and $z_0, ..., z_{k-1} \neq z$.

In other words, $Delay[(z_i), z]$ is the total time units after which the state $z$ on the path $(z_i)$ is reached at the first time, *i.e.*, the minimal time distance from $z_0$ to $z$. Let $z_0 \in Z$ be a state of the reachability graph and $\varphi$ be a TCTL formula. Then the relation $\models$ for TCTL is defined as follows:

- $z_0 \models EF[l, h]\varphi$, if there is a path $(z_i)$ and $j > 0$ such that $z_j \models \varphi$ and $l \leq Delay((z_i), z_j) \leq h$, where $[l, h]$ is a time interval of natural numbers with $0 \leq l < h \leq w$ ($w$ is a fixed integer),
- $z_0 \models AF[l, h]\varphi$, if for all paths $(z_i)$, there is $j > 0$ such that $z_j \models \varphi$ and $l \leq Delay((z_i), z_j) \leq h$.

# Chapter 3 Possible Reconfigurations in NCESs

Reconfigurable control systems are characterized by clear modular structure. Net condition/event control system (NCES) is such a modular formalism that was developed for modeling and analyzing industrial distributed control systems. Assume that an industrial control system is expected to be reconfigurable. It means that the controllers of its distributed physical components should be able to change themselves actively. According to changed execution environment or user requirements, these controllers should be able to be standby, activated, or even be removed from the system. In addition, they should also be able to change their connection relation with other controllers, or be able to modify their own behavior modes, or just update some shared data. If NCESs are applied to model such reconfigurable systems, components of NCESs such as places, transitions, flow arcs, and markings within particular basic modules or condition/event signals among these modules should be modified at run-time. This chapter focuses on dynamic reconfigurations and control of NCESs.

## 3.1 Motivation

The main reasons to prefer NCESs/TNCESs to many others formalisms specifying an industrial control system can be explained by two aspects. The first is their non-interleaving semantics *i.e.*, possibility of firing several transitions simultaneously, which better fits to modeling of distributed processes and of their interaction. The second is the more compact reachability space [71], [72], [73].

Generally, in an NCES, the number of tokens in places classically correspond to system states of a control system. The firing of transitions, *i.e.*, the occurrence of discrete events such as the receipt of pulse/step signals, may change the number of tokens in places. There are several conditions to be fulfilled to enable a transition to fire. First of all, all pre-places have to be marked with at least one token. In addition, it may have incoming condition arcs from places and event arcs from other transitions. A transition is enabled by condition signals if each of its source places is marked by at least one token. The other type of influence on the firing can be described by event signals that come to the transition from other transitions.

Assume that a reconfigurable control system is modeled by an NCES. Different ele-

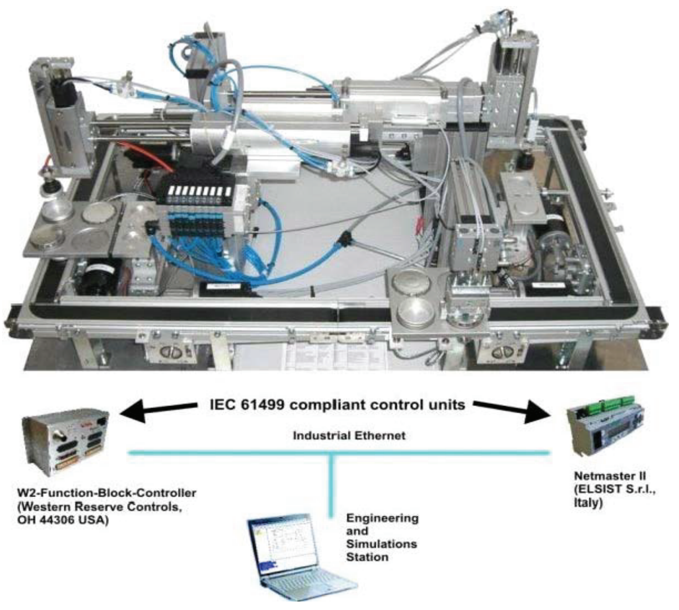

IEC 61499 compliant control units
Industrial Ethernet
W2-Function-Block-Controller (Western Reserve Controls, OH 44306 USA)
Engineering and Simulations Station
Netmaster II (ELSIST S.r.l., Italy)

Fig 3.1 EnAS demonstrator in Halle [1]

ments of the NCES model correspond to different physical/abstract parts of the reconfigurable control system. In this chapter, an automatic reconfiguration of an NCES means any addition/removal of places, transitions, and any update in the initial marking. Meanwhile, any modification of them should be well controlled to avoid failure such as deadlock and overflow.

As far as we know, there is no research work dealing with automatic reconfigurations based on NCESs, where relevant studies by using other formalisms are investigated in Chapter 1. This chapter first builds an NCES-based model for a potential reconfigurable system. After that, possible reconfiguration scenarios are defined. Then, a special mechanism is proposed to handle these reconfiguration scenarios automatically. Three types of NCES-based modules are defined. The first module allows the addition/removal of places to/from the system NCES model. The second module handles the addition/removal of transitions to/from the system within the given subset of places. The third one copes with modifications of the initial marking. Finally, in order to guarantee a safe behavior of this reconfigurable architecture, a model checking for the verification of CTL-based properties is applied. The whole work is illustrated by an experimental manufacturing platform.

## 3.2 Experimental Manufacturing Platform

An experimental manufacturing platform EnAS, as shown in Fig. 3.1, was designed as a prototype to demonstrate this work. It is supposed to have the following behavior: it

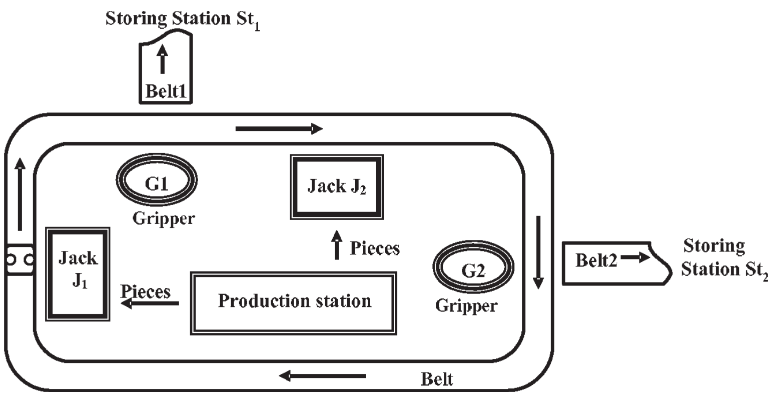


Fig 3.2 Working process of EnAS

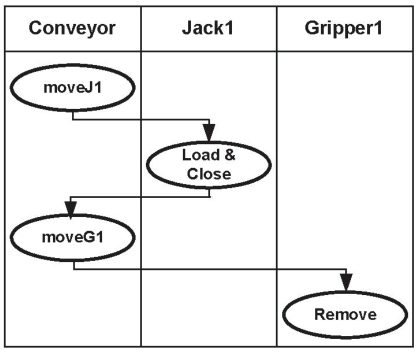


Fig 3.3 Policy 1: Production of a tin

transports pieces from a previous production system (Detailed descriptions are available in the Website of the research laboratory of Prof. H. M. Hanisch at Martin Luther University Halle-Wittenberg (http:// aut.informatik.uni-halle.de) into storing units. The pieces in EnAS shall be placed inside tins to close with caps afterwards. Two different production strategies can be applied: one or two pieces should be correctly placed in each tin according to production rates of pieces of tins and caps. We denote respectively by $nb_{pieces}$, $nb_{tins+caps}$ the production number of pieces and tins (as well as caps) per hour and by $Threshold$ a variable(defined in user requirements) to choose the adequate production strategy.

The EnAS system is mainly composed of a belt, two Jack stations (J1 and J2) and two Gripper stations (G1 and G2) (Fig. 3.2). The Jack stations place new produced pieces and close tins with caps, whereas the Gripper stations remove charged tins from the belt into the storing units. Initially, the belt moves a particular pallet containing a tin and a cap into the first Jack station J1. According to production parameters, we distinguish two cases (see Fig. 3.3 and 3.4):

- **First production policy**: If ($nb_{pieces}/nb_{tins+caps} \leq Threshold$), then the Jack station

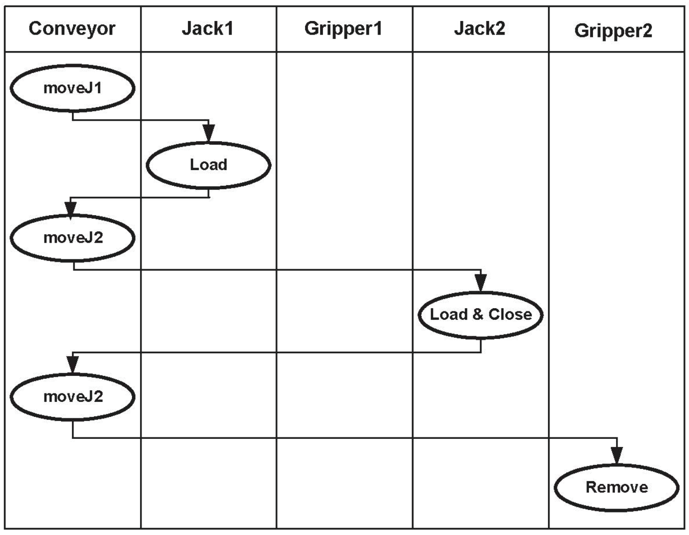


Fig 3.4 Policy 2: Production of tins with double pieces

J1 places a new piece into the tin and then closes the tin with the cap. In this case, the Gripper station G1 removes the tin from the belt into the storing station St1 (Fig. 3.3).

- **Second production policy**: If ($nb_{pieces}/nb_{tins+caps} > Threshold$), then the Jack station J1 places just a piece in the tin which is moved thereafter into the second Jack station to place a second new piece. Once J2 closes the tin with a cap, the belt moves the pallet into the Gripper station G2 to remove the tin (with two pieces) into the second storing station St2 (Fig. 3.4).

## 3.3 Specification of Reconfigurable Control Systems

Reconfiguration means qualitative changes in structures, functionalities, and algorithms of control systems as responses to qualitative changes of goals of controls, of controlled systems, or of environments the systems behaves within. This could be caused by (partial) failures, breakdowns, or even by human interventions. Let us denote by $Sys$ the reconfigurable control system to be modeled by NCESs $\Sigma(Sys)$ that specify all possible behaviors of the system to be applied after well-defined reconfigurations.

$$\Sigma(Sys) = \{P_{\Sigma(Sys)}, T_{\Sigma(Sys)}, F_{\Sigma(Sys)}, CN_{\Sigma(Sys)}, EN_{\Sigma(Sys)}, V_{\Sigma(Sys)}, m0_{\Sigma(Sys)}\}$$

We mean by a reconfiguration scenario of $\Sigma(Sys)$ 1) any addition/removal of places, 2) any addition/removal of transitions, and 3) any update of marking. The system can be specified by different sub-NCESs defining different possible behaviors to be followed under

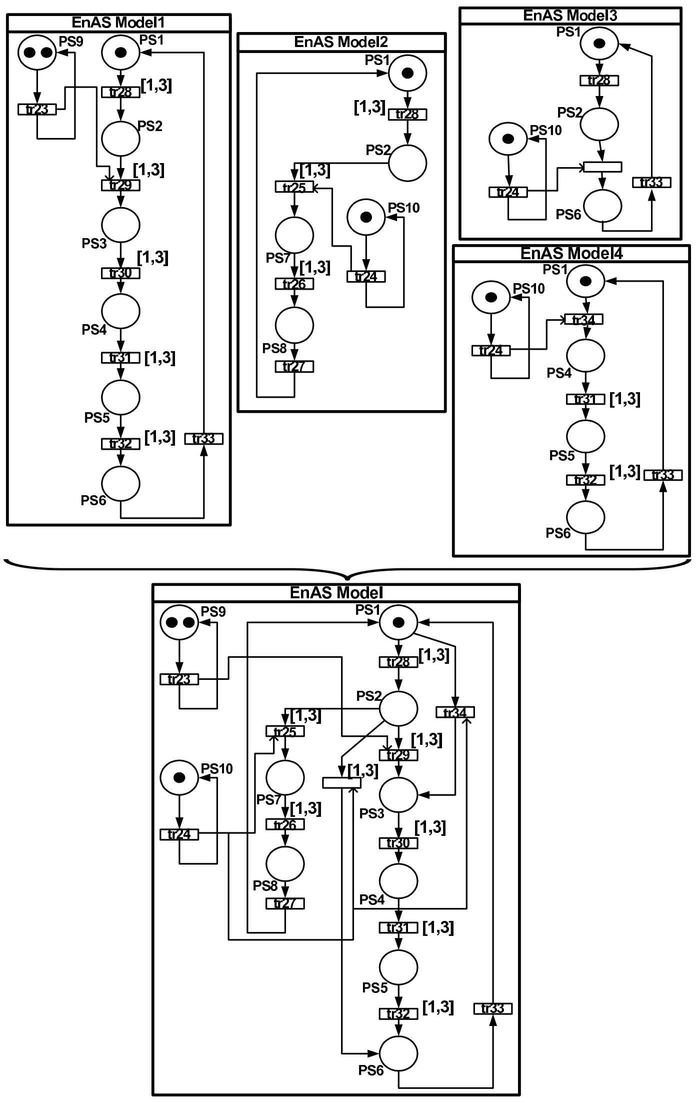


Fig 3.5 Specification of the reconfigurable EnAS with NCESs

well-defined conditions. Let $\xi(Sys)$ be a sub-NCES that models $Sys$ after a well-defined automatic reconfiguration scenario. Here a sub-NCES means the NCES-based model of a configuration of the reconfigurable system $Sys$.

$$\xi(Sys) = \{P_{\xi(Sys)}, T_{\xi(Sys)}, F_{\xi(Sys)}, CN_{\xi(Sys)}, EN_{\xi(Sys)}, V_{\xi(Sys)}, m0_{\xi(Sys)}\},$$

where, $P_{\xi(Sys)} \subseteq P_{\Sigma(Sys)}$, $T_{\xi(Sys)} \subseteq T_{\Sigma(Sys)}$, and $F_{\xi(Sys)} \subseteq F_{\Sigma(Sys)}$.

If $\xi(Sys)$ specifies the configuration when a particular reconfiguration scenario is applied to the system, the places of $P_{\xi(Sys)}$ (resp, transitions of $T_{\xi(Sys)}$ and arcs of $F_{\xi(Sys)}$) become the only able places of $P_{\Sigma(Sys)}$ to be activated (resp, only able transitions of $T_{\Sigma(Sys)}$ and able arcs of $F_{\Sigma(Sys)}$). The rest of places, transitions and arcs become disable.

In the Benchmark Production System EnAS, only four sub-NCESs are possible to specify its behavior when well-defined reconfiguration scenarios are automatically applied at run-time, as shown in Fig. 3.5. To make it clear, the four sub-NCESs are respectively shown above, where $\Sigma(Sys)$ is graphically shown below.

- Let $\xi_1(Sys)$ be the first sub-NCES that specifies EnAS when the Second Production Policy is applied such that:

  $$P_{\xi_1(Sys)} = \{PS1, PS2, PS3, PS4, PS5, PS6, PS9\}$$

  Place $PS1$ corresponds to the displacement of an empty tin on the belt to the first Jack station where a piece is put (e.g. the place $PS2$). The tin is displaced thereafter (e.g. place $PS3$) to the second Jack station where a second piece is put before it is closed with a cup (e.g. place $PS4$). The closed tin is displaced thereafter on the belt (e.g. place $PS5$) to the second Gripper station $G2$ for an evacuation to the second storing station $St2$. We note finally that place $PS9$ defines the number of pieces (e.g. two pieces) to be put in the tin when the Second Production Policy is applied.

- Let $\xi_2(Sys)$ be the second sub-NCES that specifies EnAS when the First Production Policy is applied such that:

  $$P_{\xi_2(Sys)} = \{PS1, PS2, PS7, PS8, PS10\}$$

  Place $PS7$ corresponds to the displacement of a tin containing a piece and closed with a cup from the first Jack station to the first Gripper station (e.g. place $PS8$). We note finally that the place $PS10$ defines the number of pieces (e.g. one piece) to be put in the tin when the First Production Policy is applied.

- Let $\xi_3(Sys)$ be the third sub-NCES that specifies EnAS when the second Jack station is broken such that:

  $$P_{\xi_3(Sys)} = \{PS1, PS2, PS6, PS10\}$$

  Place $PS2$ corresponds to the placement of a piece in a tin to be closed with a cup in the first Jack station. The place $PS6$ corresponds to the removal from the belt to the second Storing Station $St2$.

- Let $\xi_4(Sys)$ be the fourth sub-NCES that specifies EnAS when the first Jack station is broken such that:

$$P_{\xi_4(Sys)} = \{PS1, PS3, PS4, PS5, PS6, PS10\}$$

  Places $PS1$ and $PS3$ correspond to the displacement of an empty tin on the belt to the second Jack station where a piece and a cup are put (e.g. the place $PS4$). The closed tin is displaced thereafter on the belt (e.g. place $PS5$) to the second Gripper station $G2$ for an evacuation to the second storing station $St2$ (e.g. $PS6$). We note finally that the place $PS10$ defines the number of pieces (e.g. only one piece) to be put in the tin when the first Jack station is broken.

## 3.4 Reconfiguration of Net Condition/Event Systems

To dynamically reconfigure the NCES $\Sigma(Sys)$, we define nested state machines where states correspond to other state machines. Each state machine forms a module allowing reconfigurations of the system. Three types of modules are distinguished:

- The first module called $changer\_places$ is modeled by an NECS to be denoted by $CP$ in which each place $p$ = $reconfigure(\xi(Sys))$ corresponds to a subset $P_{\xi(Sys)}$ $\subseteq P_{\Sigma(Sys)}$. Therefore each transition in this state machine corresponds to the addition/removal of places to/from the system's specification. $CP$ is formalized as $CP$ = $\{P_{CP}, T_{CP}, F_{CP}, CN_{CP}, EN_{CP}, V_{CP}, m0_{CP}\}$.

- For each place $p$ of $CP$, we define a particular module called $changer\_transitions$ and modeled by an NCES to be denoted by $CT$ ($CT$= $transition(p)$) in which each place corresponds to a particular composition of places in the system's specification $\xi(Sys)$. Each transition corresponds therefore to the addition/removal of transitions, event/condition signals in $\xi(Sys)$ ($p$ = $reconfigure(\xi(Sys))$). $CT$ is formalized as $CT$ = $\{P_{CT}, T_{CT}, F_{CT}, CN_{CT}, EN_{CT}, V_{CT}, m0_{CT}\}$.

- The third particular type of modules is called $changer\_marking$. It is modeled by an NCES and is denoted by $CM$. In $CM$, each place corresponds to a particular marking of $\Sigma(Sys)$. A place of $CM$ corresponds to one or more places of a module $changer\_transitions$ or the whole module $changer\_places$. $CM$ is formalized as $CM$ = $\{P_{CM}, T_{CM}, F_{CM}, CN_{CM}, EN_{CM}, V_{CM}, m0_{CM}\}$.

We denote by $\Delta(CT)$ (resp. $\Delta(CM)$) the set of $CT$ (resp. $CM$) modules. The whole control system is characterized by different behaviors such that each one should be executed after a well-defined reconfiguration scenario. Each scenario to be denoted by $(p, q, k)$ ($p \in P_{CP}$, $q \in P_{CT}$ = $transition(p)$ such that $CT \in \Delta(CT)$, and $k \in P_{CM}$ such that $CM \in \Delta(CM)$) is executed when the corresponding place $p$ is active in $CP$, place $q$ is active in $CT$ and finally place $k$ is active in the module $CM$. We denote by $Behavior_{p,q,k}(Sys)$ the sub-NCES of $\Sigma(Sys)$ that can implement $Sys$ when the reconfiguration scenario $(p, q, k)$ should be automatically applied. We synchronize the modules $CP$, $CT$ and $CM$ by event signals as follows: For each scenario $(p, q, k)$,

- $\forall t1 \in^{\bullet} p$ and $t2 \in^{\bullet} q, \exists ev1 \in (t1, t2)$,
- $\forall t2 \in^{\bullet} q$ and $t3 \in^{\bullet} k, \exists ev2 \in (t2, t3)$.

We synchronize in addition the reconfiguration modules and the specification $\Sigma(Sys)$ of the system $Sys$ by event signals as follows: For each scenario $(p, q, k)$ such that $Behavior$ $_{p,q,k}$ $(Sys) = \xi(Sys)$,

- $\forall t1 \in^{\bullet} q, \exists t2 \in T_{\xi(Sys)}$ such that $\exists ev1 = (t1, t2)$,
- $\forall t3 \in^{\bullet} k, \exists t4 \in T_{\xi(Sys)}$ such that $\exists ev2 = (t3, t4)$.

The events $ev1$ and $ev2$ allow applications of reconfiguration scenarios to activate places and/or transitions and/or arcs and/or to change marking in the NCES $\xi(Sys) \in \Sigma(Sys)$.

**Example 1** *The final NCES-based model of the reconfigurable experimental manufacturing system EnAS is shown in Fig. 3.6. According to Fig. 3.3, the module $Changer\_places$ $CP1$ is composed of two places $P1$ and $P2$ that respectively define the Second and the First Production Policy. The transitions $tr1$ and $tr2$ define in this case the addition and removal of places in the system's specification. When transition $tr1$ is fired, we disable the places $PS3$, $PS4$, $PS5$, $PS6$ and $PS9$, and we activate the places $PS7$, $PS8$ and $PS10$. We associate for the place $P1$ the NCES $CT1$ and for the place $P2$ the NCES $CT2$. The place $P4$ of the module $CT1$ corresponds to the execution of the second production policy when $PS1, PS2, PS3, PS4, PS5, PS6, PS9$ are specifying EnAS. The place $P5$ specifies the system when the second Jack station is broken. The place $P6$ corresponds to any problem in the first Jack station. The place $P7$ is reached when the first and the second Jack stations*

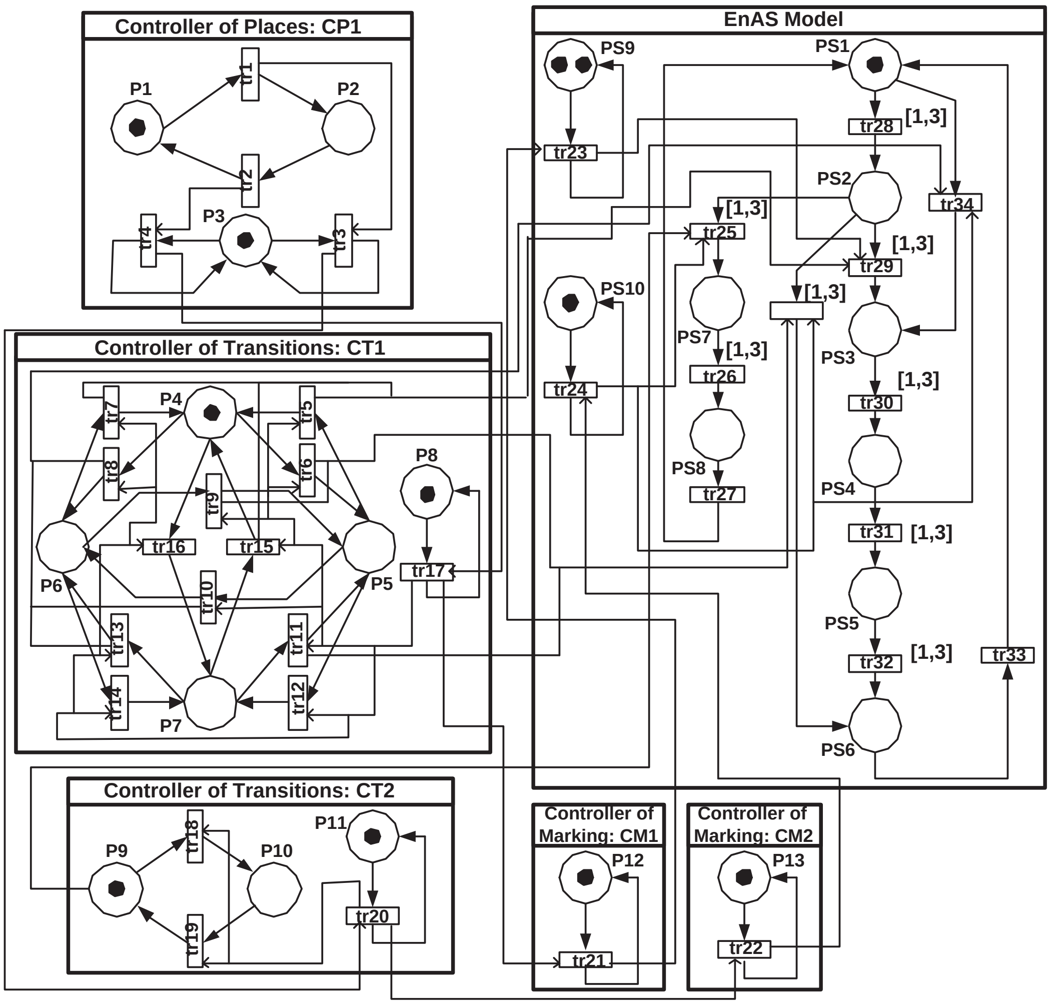


Fig 3.6 NCES-based modules for automatic reconfigurations of EnAS

*are broken. The place $P9$ of the module $CT2$ defines an execution scenario of EnAS when the first Jack and Gripper stations are used to produce pieces. We note in addition that the places $P12$ is active from the module $CT1$ when we put two pieces in the tin, whereas the place $P13$ is active when only one piece is put in the tin (e.g. it is activated by $CT2$).*

## 3.5 System Verification

Once the reconfigurable NCES is well-modeled, the next step to be addressed is their verification in order to guarantee a correct behavior of the system after implementation of any reconfiguration scenario. In this dissertation, we use the model checker SESA to verify CTL-based properties defined in user requirements. This tool allows the verification of any reactions of reconfiguration modules as well as their synchronization with the system's NCES that should be checked too. We show in Fig. 3.7 a reachability graph generated by

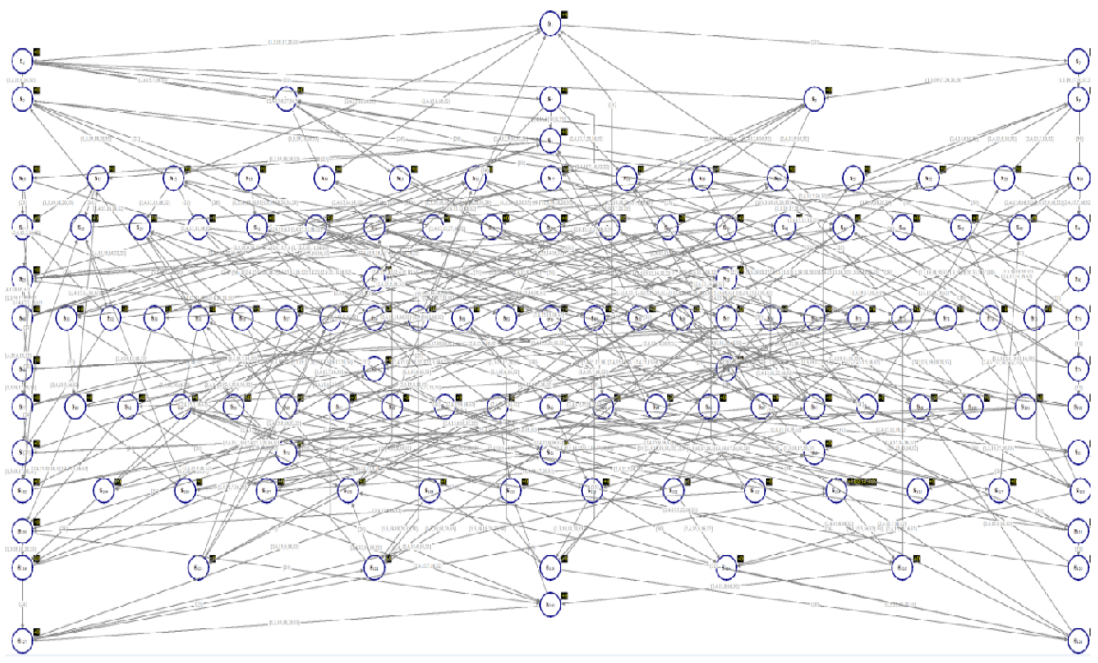


Fig 3.7 Reachability graph of the reconfigurable architecture

SESA for the verification of the NCES depicted in Fig. 3.6.

**Example 2** *In the system EnAS, we check functional properties of the NCES-based state machines and the system's NCES. We have to check in particular that whenever the transition* $tr1$ *is fired, then the place* $PS7$ *should be reached:*

$$AGAtr1XPS7$$

*This formula is proven to be True by applying this tool. Indeed, when conditions are satisfied to apply the Second Production Policy, the state PS7 should be reached. We have also to check that whenever the transition* $tr5$ *is fired to apply the second policy, the place* $PS5$ *should be applied to bring the tin from the first and second Jack stations to the second Gripper station:*

$$AGAtr5XPS5$$

*This formula is proven to be True. We check also the correct behavior of the system EnAS when the Second Production Policy is applied by verifying the following formula:*

$$AGAtr29XAFEtr30XAFEtr31XAFEtr32XTRUE$$

*Indeed, whenever the belt is activated to transport a piece to the first Jack station, it is activated again to transport the piece to the second Jack station before reaching the second Gripper station. This formula is proven to be True by SESA. When the Second Production Policy is applied, we check also if the evacuation of a closed tin from the belt can be done in 4 time units. The following formula is proven to be False by SESA:*

$$EF[3,4]PS6$$

*The following formula is proven to be True:*

$$AF[5,6]PS6$$

*Indeed the state $PS6$ (e.g. evacuation from the belt) should be reached 5 time units at least after the activation of the place $PS1$.*

## 3.6 Summary

This chapter deals with automatic reconfigurations to dynamically change the behaviors of control systems, which is enabling or disabling certain parts of the system. This is a new challenge in industry. We specify such systems with reconfiguration behavior by NCESs that are an extension of Petri nets. Herein, a reconfiguration scenario is any addition-removal-update of places, transitions, or just the modification of the initial marking. We define formal modules allowing reconfigurations of an NCES, where the first module deals with places, the second with transitions and the third with the marking. We apply a model checking for the verification of CTL-based functional properties in order to guarantee a safe behavior of this reconfigurable architecture.

# Chapter 4 Reconfigurable Timed Net Condition Event Systems

From previous chapter, we notice that if Net Condition/Event Systems (NCESs) are applied directly to model a reconfigurable discrete event control system (RDECS). The system will be enlarged greatly due to new controllers designed for different reconfiguration scenarios. This sharply burdens the verification of the final system. Therefore, a new formalism Reconfigurable Timed Net Condition/Event System (R-TNCES) is proposed for the modeling and verification of RDECSs. This chapter represents the motivation of the R-TNCESs, basic definitions and properties of an R-TNCES, and a verification method for a particular type of R-TNCESs.

## 4.1 Motivation

A reconfiguration scenario is applied for the automatic improvement of the system performance or for the system protection at run-time when hardware faults occur [56], [74]. The good performance of an RDECS should be that it can reconfigure automatically and rapidly due to the changed environment or user requirements without a halt [75]. In addition, during a reconfiguration process, the internal behavior of the components in the working environment should not be influenced and no deadlock arises [76], [77], [78]. These advanced requirements and the conspicuous extra complexity throw the industry and academia new challenges to develop RDESs [79], [80].

As shown in Chapter 1, many researchers have tried to deal with the modeling of control systems with potential reconfigurations. Most of them can describe the system's reconfiguration behavior. However, some of them do not clearly define the modularity, which brings complexity in designing, understanding, and future redeveloping. The system's correctness such as coherence of states before and after system reconfigurations is not considered. Moreover, temporal constraints are not mentioned, which are of importance in real-time systems. Most important is that none of them deal with the reconfigurations based on the TNCES formalism. Furthermore, most of the existing methods cannot represent an RDECS in a compact manner. In this chapter, we try to model an RDECS using a direct method by defining a new formalism.

In [56], [81], a "control component" defined as a software unit was developed for the automatic refinement-based identification, specification, and verification of a plant system.

The possible reconfigurations of an NCES were studied in [1], where the modifications of places, transitions, and initial markings are handled by an agent specified by an NCES. To be independent of any approach or technology, to cover more forms of reconfiguration scenarios that can be applied at run-time such as the modification of condition/event signals in a TNCES, and to optimize the functional and temporal specification of RDECSs, a new formalism namely Reconfigurable Timed Net Condition/Event System (R-TNCES) is proposed in this chapter. A possible system configuration of an RDES is assumed as a set of physical processes with precedence constraints as done in [74]. The controllers of physical processes are modeled by control components that are specified by TNCESs with uniform interfaces through which they read data from sensors and send signals to activate actuators.

An R-TNCES is a new approach to adapt TNCESs for RDECSs such that all reconfiguration scenarios including the addition/removal of places, transitions, arcs, initial markings, and condition/event signals are specified directly. An R-TNCES consists of a behavior module and a control module. The former is composed of various control components, whose combinations form a set of superposed TNCESs that are used for the representations of certain control models of an RDES. The latter is a set of reconfiguration functions. A reconfiguration function deals with the automatic transformations of the TNCESs in response to the changes caused by errors in the controlled system, or by user requirements via enabling/disabling control components, changing condition/event signals among them, and treating the state feasibility before and after reconfigurations such that the correctness of the system can be guaranteed. The dynamic properties and implementation of an R-TNCES are illustrated in detail in the chapter. Compared with relevant studies, less extra places and transitions are needed to describe the dynamic behavior of an RDECS with the R-TNCES formalism, while more reconfiguration scenarios are covered and the temporal constraints are considered. In addition, the distinct modular structure of R-TNCESs, especially the application of control components, makes the model understandable for future extensions. A benchmark production system FESTO MPS [81] is applied to show the advantage of the R-TNCES.

Several studies have been done in recent years to control the complexity of system verification by applying hierarchical or refinement-based approaches [32], [67]. Nevertheless, no much attention is paid to TNCESs or reconfiguration forms such as the modification of condition/event signals that can be applied at run-time. R-TNCESs are an extension to TNCESs and can show a group of superposed TNCESs as well as their dynamic transformations. A verification method for R-TNCESs is necessary. SESA [62] is an effective model-checker for TNCESs, which computes the reachable states exactly. It allows the verification and analysis of the properties such as boundedness and liveness. In addition, temporal/functional properties based on Computation Tree Logic (CTL) [82], extended Computation Tree Logic (eCTL) [70], and timed Computation Tree Logic (TCTL) [58] specified by users can be checked manually.

In this chapter, an RDES is defined as a set of physical processes. The behavior of a configuration is described by several simultaneous chains that are sequences of physical processes with clear temporal constraints, where no resource competition exists among them. The controllers of the physical processes are modeled by control components. Therefore, each TNCES of an R-TNCES corresponding to the control model of a configuration is composed of a set of control components. To satisfy user requirements, the initial TNCES of an R-TNCES is checked first. Its control components are divided into multi-layers in terms of the temporal constraints and then checked layer by layer. An abstract model denoting the external environment of the underlying layer is constructed during the process, and a composite net composed of a layer and the abstract model is obtained. The reachable states of the obtained composite net are computed by SESA. Meanwhile, the functional and temporal properties based on CTL/eCTL/TCTL are checked manually. Therefore, only the composite net with a much smaller size rather than the whole TNCES is tackled at each step. Other TNCESs of an R-TNCES can be obtained one by one by implementing certain reconfiguration functions. Their verification is based on the correctness of their previous TNCESs because two TNCESs linked directly by a reconfiguration function are supposed to be very similar. If the external environment of the unchanged parts of a TNCES are not changed by implementing the reconfiguration function, the repetitive verification of the unchanged parts can be avoided. The method controls the complexity of model-checking of R-TNCESs and is applied to FESTO MPS to show its virtue.

## 4.2 Experimental Manufacturing Platform

A benchmark production system FESTO MPS as shown in Fig. 4.1 is used as an intact running example in this chapter. It is a well documented laboratory system used by many universities for research and education purposes.

The whole schematic working process of FESTO MPS together with the time cost for each physical process is shown in Fig. 4.2. It is composed of three units: the distribution unit, the test unit, and the processing unit. The distribution unit consists of two components: a pneumatic feeder and a converter. It forwards cylindrical workpieces from a stack to the

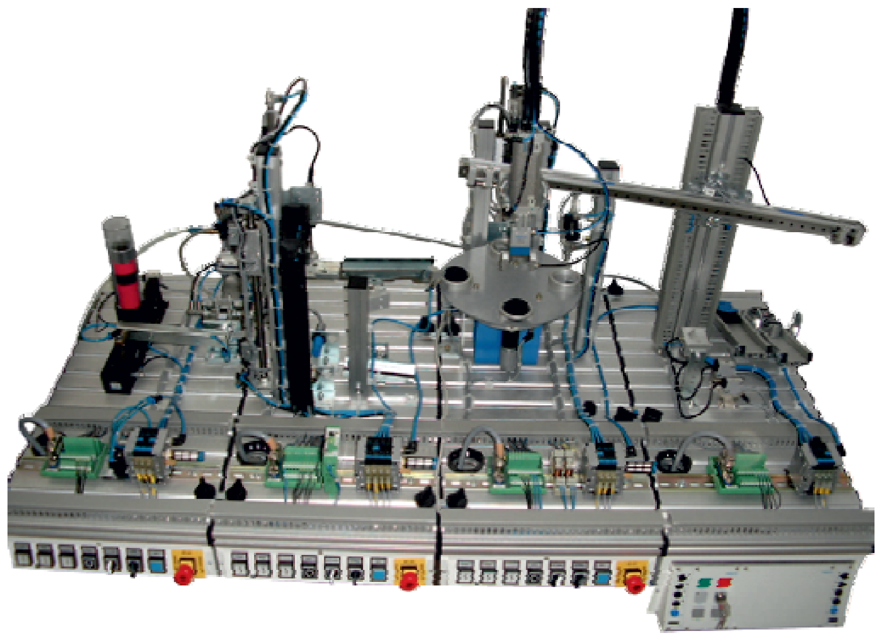


Fig 4.1 FESTO MPS

testing unit. The test unit consists of three components: the detector, the tester, and the evacuator. It performs the checking of workpieces for their height, material type, and color. Workpieces that pass the test unit successfully are forwarded to the rotating disk of the processing unit, where the drilling of workpieces is done. It is assumed in this work that there exist two drilling machines $Drill_1$ and $Drill_2$ to drill workpieces. The result of the drilling operation is next checked by a checker and finally the finished product is removed from the system by an evacuator. The set of chains describing FESTO MPS is provided as follows:

- $chain_1 = act_1, act_2, act_3, act_4,$

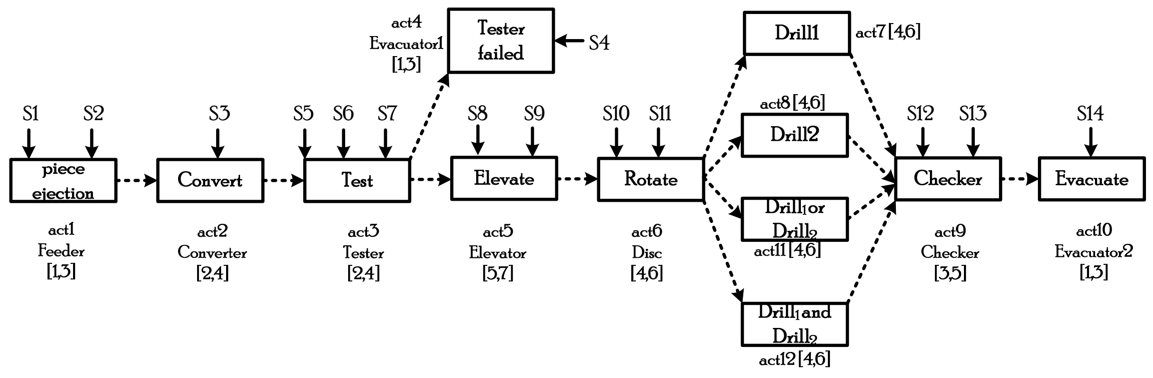


Fig 4.2 Working process of FESTO MPS

- $chain_2 = act_1, act_2, act_3, act_5, act_6, act_7, act_9, act_{10}$,
- $chain_3 = act_1, act_2, act_3, act_5, act_6, act_8, act_9, act_{10}$,
- $chain_4 = act_1, act_2, act_3, act_5, act_6, act_{11}, act_9, act_{10}$, and
- $chain_5 = act_1, act_2, act_3, act_5, act_6, act_{12}, act_9, act_{10}$.

The first chain $chain_1$ describes the system behavior that workpieces fail in passing the $Test$. $chain_2$ (*resp*, $chain_3$) describes the system behavior that workpieces pass the $Test$ before they are drilled by the drilling machine $Drill_1$ (*resp*, $Drill_2$). $chain_4$ represents the system behavior that $Drill_1$ or $Drill_2$ is used to drill workpieces when either of them is ready. The last one $chain_5$ implies that both $Drill_1$ and $Drill_2$ are used at the same time to accelerate the production. In this case, the distribution and the testing units have to forward two successive pieces to the rotating disk before starting the drilling with $Drill_1$ and $Drill_2$.

Four distinct combinations of these chains cover three behavior modes of FESTO MPS with three exclusive production rates. The light production mode is denoted by $Light_1$ (*resp*, $Light_2$) to be described by the combination of $chain_1$ and $chain_2$ (*resp*, $chain_1$ and $chain_3$), where $Drill_1$ (*resp*, $Drill_2$) is applied only, *i.e.*, $\sum chain_{L1} = \{chain_1, chain_2\}$, $\sum chain_{L2} = \{chain_1, chain_3\}$. In fact, after the execution of $Test$, a workpiece is moved to $Evacuator1$ or $Elevator$ according to the test result. $Light_1$ is the default initial behavior mode. It can be transformed into $Light_2$ while $Drill_1$ breaks down during run-time. The combinations of $chain_1$ with $chain_4$ and $chain_1$ with $chain_5$ represent the medium production mode $Medium$ and high production mode $High$ of FESTO MPS, respectively, *i.e.*, $\sum chain_M = \{chain_1, chain_4\}$ and $\sum chain_H = \{chain_1, chain_5\}$. The system completely stops in the worst case that both $Drill_1$ and $Drill_2$ are broken.

The set of control chains describing FESTO MPS' control system is presented as follows:

- $Control_{chain}1 = CC_1, CC_2, CC_3, CC_4$,
- $Control_{chain}2 = CC_1, CC_2, CC_3, CC_5, CC_6, CC_7, CC_9, act_{10}$,
- $Control_{chain}3 = CC_1, CC_2, CC_3, CC_5, CC_6, CC_8, CC_9, act_{10}$,
- $Control_{chain}4 = CC_1, CC_2, CC_3, CC_5, CC_6, CC_{11}, CC_9, CC_{10}$, and
- $Control_{chain}5 = CC_1, CC_2, CC_3, CC_5, CC_6, CC_{12}, CC_9, CC_{10}$.

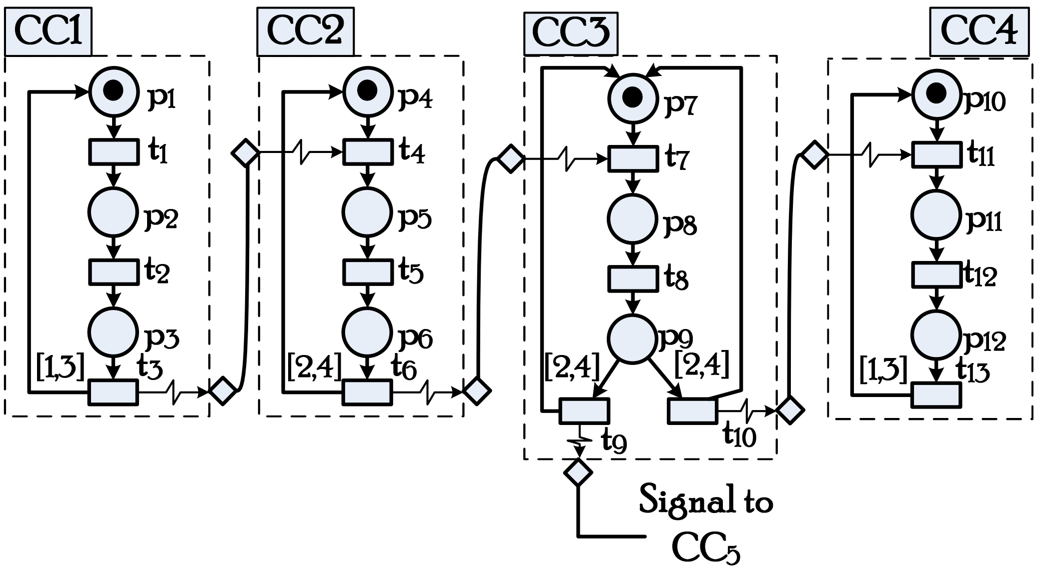


Fig 4.3 Control model of $Control_{chain}1$

Therefore, the corresponding controllers of the four configurations are denoted by

- $\sum Control_{chain}L1 = \{Control_{chain}1, Control_{chain}2\}$,
- $\sum Control_{chain}L2 = \{Control_{chain}1, Control_{chain}3\}$,
- $\sum Control_{chain}M = \{Control_{chain}1, Control_{chain}4\}$,
- $\sum Control_{chain}H = \{Control_{chain}1, Control_{chain}5\}$.

The TNCES model of $Control_{chain}1$ is graphically shown in Fig. 4.3. Actually, the control of $Feeder$, $Convert$, $Test$, and $Evacuator1$ is done one by one. Only after the control of the former physical process is finished, the latter can be activated to work.

According to user requirements, we should make the control system of FESTO MPS able to reconfigure automatically at run-time in response to any changed working environment caused by errors or new requirements to improve system performance without a halt. It is assumed that only light and medium production modes are interchangeable, so are medium and high production modes, as shown in Fig. 4.4. The high production mode can be transformed into light production mode directly, but the converse is inadmissible.

## 4.3 Control Components

The proposed formalism R-TNCES is based on "control components". The provided verification method of R-TNCESs is subjected to a specific modeling methodology. This

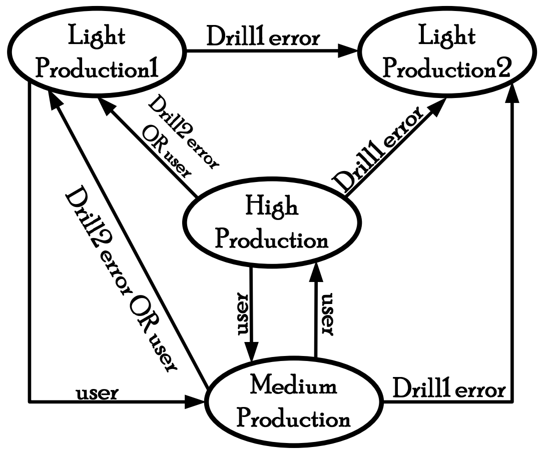


Fig 4.4 Allowed reconfigurations of FESTO MPS

section mainly introduces the concept of control components and the modeling methodology of the configurations in an RDECS.

An RDES is defined as a set of physical processes. Different compositions of these physical processes with various constraints form a family of configurations. The behavior of a configuration to be denoted by $Sys_i$ is described by a set of simultaneous chains denoted by $\sum chain_i$, where each chain is a finite sequence of actuators with clear temporal constraints. We denote the actuator of the $i^{th}$ physical process in an RDES by $act_i$, denote the time cost for the physical process to finish its work by $running(act_i) = [l_i, h_i]$, and denote the set of actuators that the system has to activate just before the activation of $act_i$ by $prev(act_i)$, where $[l_i, h_i]$ is a time interval of natural numbers with $0 \le l_i < h_i \le w$ ($w$ is a fixed integer). A chain is defined as follow:

$$chain : act_1, act_2, ..., act_n,$$

where $\forall i \in [1, n-1]$, $act_i \in pre(act_{i+1})$. It is assumed that no loop exists in a chain and no resource competition exists among the chains in $\sum chain_i$.

A control component is defined as a software unit. It is implemented by algorithms that support functionalities of a physical system and interact with its physical processes as follows:

- It reads data from a subset of sensors of the system.
- An algorithm corresponding to these sensor data is executed.
- When the execution finishes, the corresponding controlled actuators are activated.

We use control component $CC_i$ to model the controller of the $i^{th}$ physical process. A control component $CC_i$ is specified by a TNCES module that has one initial place only and is characterized by a set of traces such that each trace $tr$ is described by the following transitions:

- $t^{cc_i}_{entrance}$ is called the entrance transition. It is from the initial place. $CC_i$ is activated only after $t^{cc_i}_{entrance}$ fires. Note that each control component has one entrance transition only.
- $t^{cc_i}_{starting}$ is called a starting transition. The controlled physical process is forced to work only after it fires.
- $t^{cc_i}_{end}$ is called an end transition. An end transition fires only after the controlled physical process finishes its work and sends an event signal to the next control component $CC_{i+1}$.

In addition, a time interval is assigned to each output arc from place $p$ ($p\in{}^{\bullet}t^{cc_i}_{end}$) to $t^{cc_i}_{end}$ by $running(act_i)$. For the other output arcs, the time intervals are default.

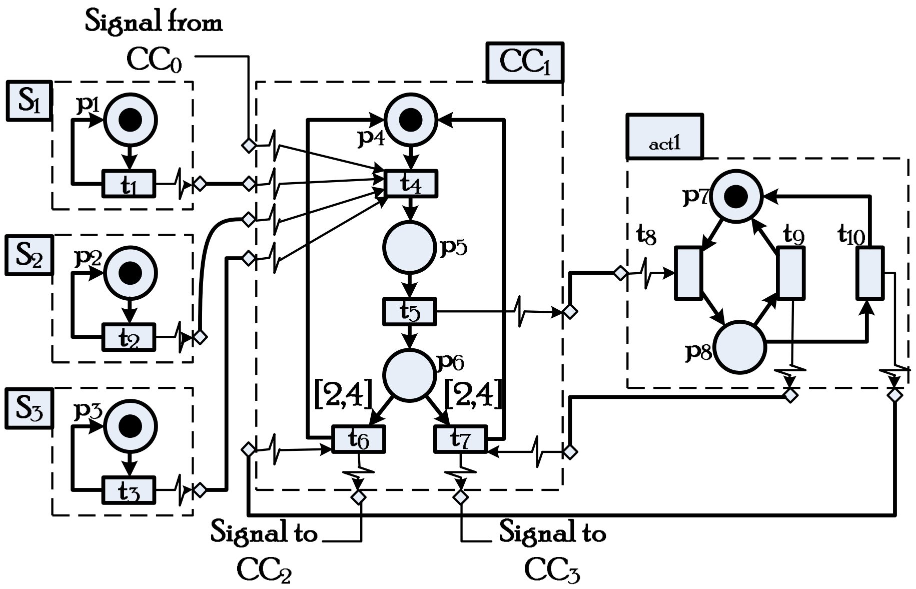


Fig 4.5 Controller model of a physical process

**Example 3** *A control component together with three sensors and an actuator is graphically shown in Fig. 4.5. The control component $CC_1$ reads data from three sensors $S1$, $S2$, and*

*$S3$, and then send signals to the actuator $act_1$ to drive the controlled physical process. $CC_1$ has two end transitions $t_6$ and $t_7$. Firing $t_6$ implies that a signal is sent to $CC_2$. Whereas, firing $t_7$ implies that a signal is sent to $CC_3$.*

According to the above definition of chains describing system behavior, the controller of a chain, namely a control chain, is defined accordingly as follows:

$$Control_{chain} : CC_1, CC_2, ..., CC_n,$$

where $CC_i$ is the controller of the $i^{th}$ physical process, which is a control component. The set of control components that the control system has to activate just before the activation of $CC_i$ is denoted by $prev(CC_i)$, where $\forall i \in [1, n-1]$, $CC_i \in pre(CC_{i+1})$. Therefore, the controller of a configuration $Sys_i$ can be described by a set of control chains denoted by $\sum Control_{chain}i$, where $|\sum chain_i| = |\sum Control_{chain}i|$.

In this chapter, it is supposed that an RDES can perform finite configurations such that the same amount of control modes are available in its corresponding RDECS. Each control mode is specified by a set of control chains, each of which is a finite sequence of control components with explicit temporal constraints. The unambiguous modularity of control components and their uniform interfaces make the control components suitable for modeling of RDECS.

## 4.4 R-TNCESs

An RDECS can perform various configurations corresponding to different behavior modes of the controlled RDES. All of them are modular and extensible, and their dynamic transformations can be performed through the activing/disactiving parts of the modules and modifying the communications among them. In this chapter, a configuration of an RDECS is modeled by a TNCES that is composed of a set of control components. We extend TNCESs in order to adapt them to RDECS. Not only reconfiguration scenarios such as the addition/removal of places, transitions, and initial markings, but also the modifications of condition/event signals among different control components are covered through this extension. In addition, the state coherence before and after a reconfiguration is considered such that the correctness and safety of the RDECS are guaranteed. An R-TNCES increases the computational power of a TNCES such that the functional as well as temporal specification of an RDECS is done directly, compactly, and optimally. This section is organized as follows: Section 4.4.1 introduces the definition of an R-TNCES, its dynamic properties and implementation are provided in Section 4.4.2 and Section 4.4.3, respectively.

### 4.4.1 Definition

**Definition 1** *An R-TNCES is a structure $RTN$=($\mathcal{B}$, $\mathcal{R}$), where $\mathcal{R}$ is control module consisting of a set of reconfiguration functions $\mathcal{R} = \{r_1, ..., r_m\}$ and $\mathcal{B}$ is behavior module that is a union of multi-TNCES, represented as*

$$\mathcal{B}=(P,\ T,\ F,\ W,\ CN,\ EN,\ DC,\ V,\ Z_0),$$

*where*

- $P$ *(*resp, $T$*) is a superset of places (*resp, *transitions),*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a superset of flow arcs,*
- $W : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ *maps a weight to a flow arc,* $W(x, y) > 0$ *if* $(x, y) \in F$*, and* $W(x, y) = 0$ *otherwise, where* $x, y \in P \cup T$*,*
- $CN \subseteq (P \times T)$ *(*resp, $EN \subseteq (T \times T)$*) is a superset of condition signals (*resp, *event signals),*
- $DC : F \cap (P \times T) \rightarrow \{[l_1, h_1], ..., [l_{|F \cap (P \times T)|}, h_{|F \cap (P \times T)|}]\}$ *is a superset of time constraints on output arcs, where* $\forall i \in [1, |F \cap (P \times T)|]$*,* $l_i, h_i \in \mathbb{N}$*, and* $l_i < h_i$*,*
- $V : T \rightarrow \{\vee, \wedge\}$ *maps an event processing mode (AND or OR) for every transition,*
- $Z_0 = (M_0, D_0)$*, where* $M_0 : P \rightarrow \{0, 1\}$ *is the initial marking, and* $D_0 : P \rightarrow \{0\}$ *is the initial clock position.*

Let $\sum TN$=$P \times T \times F \times W \times CN \times EN \times DC \times V$ denote the set of all feasible net structures that can be performed by an R-TNCES. Given a TNCES $\Gamma$=($P'$, $T'$, $F'$, $W$, $CN'$, $EN'$, $V$, $DC'$, $Z'_0$), $TN(\Gamma)$=($P'$, $T'$, $F'$, $W$, $CN'$, $EN'$, $V$, $DC'$) denotes its net structure, where $TN(\Gamma) \in \sum TN$. We have $P' \subseteq P$, $T' \subseteq T$, $F' \subseteq F$, $W' \subseteq W$, $CN' \subseteq CN$, $EN' \subseteq EN$, $DC' \subseteq DC$, $\forall t \in T'$, $V'(t) = V(t)$, $Z'_0 = (M'_0, D'_0)$, and $\forall p \in P'$, $M'_0(p) = M_0(p)$ and $D'_0(p) = D_0(p)$.

**Example 4** *We use an R-TNCES to specify FESTO MPS's control system denoted by* $RTN_{FESTO}$*. FESTO MPS is composed of 12 physical processes. We build 12 control components as their controllers, respectively. According to the control chains provided for describing control behavior of FESTO MPS in Section 4.3, it can perform four configurations within three types of behavior modes in terms of the production rates, denoted by* $Light_1$*,* $Light_2$*,* $Medium$*, and* $High$*. They are formally represented as follows:*

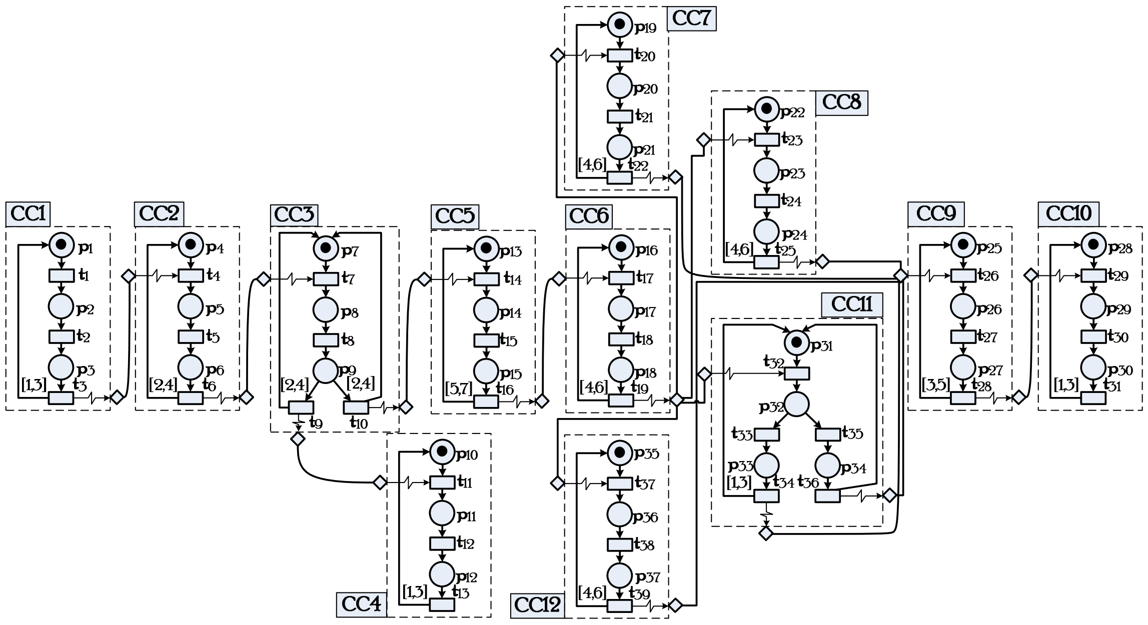


Fig 4.6 Behavior module of $RTN_{FESTO}$

- *$Light_1$=($P_1$, $T_1$, $F_1$, $W_1$, $CN_1$, $EN_1$, $DC_1$, $V_1$, $z_{01}$),*
- *$Light_2$=($P_2$, $T_2$, $F_2$, $W_2$, $CN_2$, $EN_2$, $DC_2$, $V_2$, $z_{02}$),*
- *$Medium$=($P_3$, $T_3$, $F_3$, $W_3$, $CN_3$, $EN_3$, $DC_3$, $V_3$, $z_{03}$),*
- *$High$=($P_4$, $T_4$, $F_4$, $W_4$, $CN_4$, $EN_4$, $DC_4$, $V_4$, $z_{04}$).*

*The behavior module of $RTN_{FESTO}$ is graphically shown in Fig. 4.6. The four control modes are covered by it. It is formally described as follow:*

$$\mathcal{B}_{FESTO}=(P,\ T,\ F,\ W,\ CN,\ EN,\ DC,\ V,\ Z_0).$$

*$TN(Light_1), TN(Light_2), TN(Medium), TN(High) \in \sum TN_{FESTO}$. We have $P = P_1 \cup P_2 \cup P_3 \cup P_4$, $T = T_1 \cup T_2 \cup T_3 \cup T_4$, $F = F_1 \cup F_2 \cup F_3 \cup F_4$, $W = W_1 \cup W_2 \cup W_3 \cup W_4$, $CN = CN_1 \cup CN_2 \cup CN_3 \cup CN_4$, $EN = EN_1 \cup EN_2 \cup EN_3 \cup EN_4$, $DC = DC_1 \cup DC_2 \cup DC_3 \cup DC_4$, $\forall t \in T_1 \cap T_2 \cap T_3 \cap T_4$, $V(t) = V_1(t) = V_2(t) = V_3(t) = V_4(t)$, and $\forall p \in P_1 \cap P_2 \cap P_3 \cap P_4$, $Z_0(p) = z_{01}(p) = z_{02}(p) = z_{03}(p) = z_{04}(p)$. Eight different reconfiguration scenarios can be applied to FESTO MPS as shown in Fig. 4.4. Thus the control module of FESTO MPS's R-TNCES has eight elements. It is formally described as follow:*

$$\mathcal{R}_{FESTO} = \{\ r_{L_1,L_2},\ r_{L_1,M},\ r_{M,L_1},\ r_{M,L_2},\ r_{M,H},\ r_{H,L_1},\ r_{H,L_2},\ r_{H,M}\ \}.$$

Let $^{\bullet}r$ (*resp*, $r^{\bullet}$) denote the original (*resp*, target) TNCES before (*resp*, after) a reconfiguration function $r$ is applied, where $TN(^{\bullet}r), TN(r^{\bullet}) \in \sum TN$.

**Definition 2** *A reconfiguration function r is a structure r=*$(Cond,\ s,\ x)$. $Cond \rightarrow \{true, false\}$ *is the pre-condition of r.* $s$: $TN(^{\bullet}r) \rightarrow TN(r^{\bullet})$ *is the structure modification instruction.* $x$: $last_{state}(^{\bullet}r) \rightarrow initial_{state}(r^{\bullet})$ *is the state correlation function, where* $last_{state}(^{\bullet}r)$ (resp, $initial_{state}(r^{\bullet})$) *denotes the last* (resp, *initial) state of* $^{\bullet}r$ (resp, $r^{\bullet}$) *before* (resp, *after) the application of r.*

If $Cond = true$, $r$ is executable, otherwise it cannot be executed. The structure modification instruction guides the structure modification from $TN(^{\bullet}r)$ to $TN(r^{\bullet})$, including the addition /removal of control components and condition/event signals among them such that $TN(^{\bullet}r)$ is transformed into $TN(r^{\bullet})$. The state correlation function maps the last state of $^{\bullet}r$ before the application of $r$ to a feasible initial state of $r^{\bullet}$ after the application of $r$, from which the reconfigured system goes on running.

The pre-condition of a reconfiguration function traditionally means specific external instructions, gusty component failures, or the arrival of certain states. It is assumed that all states of the controlled RDES are obtained immediately and can be used directly in this chapter. The fundamental structure modification instructions are proposed in Table 4.1. We denote by $x$ a place, $y$ a transition, $\mathbb{CC}$ a control component, and "$+$" the *AND* of instructions in order to represent complex modification instructions.

| Instruction | Symbol |
|---|---|
| Add condition signals | $Cr(cn(x, y))$ |
| Add event signals | $Cr(ev(y, y))$ |
| Add control component | $Cr(\mathbb{CC})$ |
| Delete condition signals | $De(cn(x, y))$ |
| Delete event signals | $De(ev(y, y))$ |
| Delete control component | $De(\mathbb{CC}))$ |

Table 4.1 Fundamental structure modification instructions of R-TNCESs

**Example 5** $r_{L_1,L_2}$=$(Cond,\ s,\ x)$. $Cond$ = $true$ *while* $Drill_1$ *breaks down.* $s$=$De(CC_7)$ + $Cr(CC_8)$ + $Cr(ev(t_{19}, t_{23}))$ + $Cr(ev(t_{25}, t_{26})$. *Let us assume that FESTO MPS is in the light production mode* $Light_1$. $Drill_1$ *breaks down at time* $\tau$ *while the system is at state* $z_1 = last_{state}(Light_1) = (M_1, D_1)$, *where* $M_1 = p_1 + p_4 + p_9 + p_{10} + p_{13} + p_{16} + p_{21} + p_{25} + p_{28}$, *and* $D_0 = p_4 + 4p_9 + p_{10} + p_{13} + 26p_{16} + 3p_{21} + 26p_{25} + 26p_{28}$ *(* i.e., *a* *workpiece is being tested, and a workpiece is being drilled by $Drill_1$). Then $r_{L_1,L_2}$ is executed automatically to continue the production at this time. The first step is to execute the structure modification instruction $s$, including removing $CC_7$, adding $CC_8$, and modifying the signals among $CC_8$ and other control components. According to the last state $z_1$ before $r_{L_1,L_2}$ is implemented, the system should go on working from state $z_2 = (M_2, D_2)$, where $M_2$=$p_1$+$p_4$+$p_9$+$p_{10}$+$p_{13}$+$p_{16}$+$p_{22}$+$p_{25}$+$p_{28}$, and $D_2$=$p_4$+4$p_9$+$p_{10}$+$p_{13}$+26$p_{16}$+26$p_{25}$+26$p_{28}$,* i.e., *the workpiece on $Test$ goes on being tested and $Drill_2$ is added into the system waiting for a drilling task. The controlled system continues to work in $Light_2$ without any reboot from $z_2$.*

### 4.4.2 Dynamics of R-TNCESs

The dynamics of an R-TNCES is represented in this section by referring to self modification nets and net rewriting systems. The states of an R-TNCES are defined as follows:

**Definition 3** *Let $\Gamma$ be a TNCES supported by an R-TNCES $RTN$. A state of $RTN$ is a pair $(TN(\Gamma), State(\Gamma))$, where $TN(\Gamma)$ denotes the net structure of $\Gamma$ and $State(\Gamma)$ denotes a state of $\Gamma$.*

The evolution of an R-TNCES depends on what events (reconfiguration functions or transitions) take place. Let $\Gamma$ be the current active TNCES with $\Gamma$=($P$, $T$, $F$, $W$, $CN$, $EN$, $DC$, $V$, $Z_0$) and $TN(\Gamma) \in \sum TN$. If a maximal step $u$ fires, $\Gamma$ evolves from its one inner state to another. However, if a reconfiguration function $r$ is applied, then $\Gamma$ is updated into $\Gamma'$ by changing its net structure and updating its state.

A reconfiguration function $r$=($Cond$, $s$, $x$) is enabled at state $(TN(\Gamma), State(\Gamma))$ if

1. $Cond$ = $true$, *i.e.*, its pre-condition is fulfilled;

2. $TN(\Gamma)$ = $TN({}^{\bullet}r)$, *i.e.*, $TN(\Gamma)$ is equal to the net structure of ${}^{\bullet}r$;

3. $\exists State(r^{\bullet})$, $x(State(\Gamma))$=$initial_{state}(r^{\bullet})$, *i.e.*, there exists a proper state $initial_{state}$ $(r^{\bullet})$ from which the system goes on working.

An enabled reconfiguration function can fire. After firing a reconfiguration function $r$ at state $(TN(\Gamma), State(\Gamma))$, the system evolves into a new state $(TN(\Gamma'), State(\Gamma'))$ where the system structure is modified from $TN(\Gamma)$ to $TN(\Gamma')$ and the state is updated to $State(\Gamma')$.

For a transition $t$ in an R-TNCES, the first condition of its firing is that it must be in the current active TNCES. On this basis, the firing rule of a transition in an R-TNCES is the same as in a TNCES.

A spontaneous transition $t$ is enabled at a state $(TN(\Gamma), State(\Gamma))$ with $State(\Gamma) = (M, D)$ if it has both token concession and condition concession, *i.e.*, $\forall p \in {}^{\bullet}t$, $M(p) \geq 1$

and $\forall p \in {}^{-}t$, $M(p) \geq 1$. In the case of $V(t) = \wedge$, a forced transition $t$ is enabled at the state $z = (M, D)$ if it has both token concession and condition concession, all its forcing transitions are enabled or it does not have forcing transitions, and the clock position of its each input place is within the corresponding time constraint, *i.e.*,

1. $\forall p \in {}^{\bullet}t$, $M(p) \geq 1$,
2. $\forall p \in {}^{-}t$, $M(p) \geq 1$,
3. $\forall t' \in {}^{\sim}t$, $\forall p \in {}^{\bullet}t'$, $M(p) \geq 1 \vee {}^{\sim}t = \emptyset$, and
4. $\forall p \in {}^{\bullet}t$, $eft(p,t) \leq D(p) \leq lft(p,t)$.

In the case of $V(t) = \vee$, a forced transition $t$ is enabled if it has both token concession and condition concession, at least one of its forcing transitions is enabled or it does not have forcing transitions, and the clock position of each its input place is within the corresponding time constraint. Formally, a forced transition, in this case, is enabled if the third condition above is changed into $\forall p \in {}^{\bullet}t$, $\exists t' \in {}^{\sim}t$, $M(p) \geq 1 \vee {}^{\sim}t = \emptyset$.

Spontaneous transitions can fire at each time instant as long as they are enabled. Hence, all enabled spontaneous transitions are included (and the whole set of transitions that are forced by them) into the set of firing transitions. The maximal steps are generated from the set of fired transitions. The time after which a maximal step fires is called a step delay and is denoted by $\Delta\tau$. In the case that there is no spontaneous transition, $\Delta\tau$ is equal to the earliest firing time of a forced transition. Firing a maximal step $u$ after $\Delta\tau$ time units results in a new state $z' = (M', D')$ with $M' = M + [N] \times u$ ($[N]$ is a $|P| \times |T|$ integer matrix with $[N](p,t) = W(t,p) - W(p,t)$) and $\forall p \in P$, $D(p)' = D(p) + \Delta\tau$, if $M(p) > 0 \wedge \forall t \in u, t \notin {}^{\bullet}p \cup p^{\bullet}$, otherwise $D(p)' = 0$.

In an RDECS, if an hardware error occurs or new user requirements arise at run-time, a reconfiguration scenario should be implemented. Note that in an R-TNCES, a reconfiguration function always has a higher priority than a transition. That is to say, if a reconfiguration function $r$ and a transition $t$ are enabled simultaneously at a state $z$, the reconfiguration function always fires first.

**Definition 4** *The reachability graph of the R-TNCES $RTN$ is a combination of several labeled directed graphs whose nodes are states of $RTN$ and whose arcs are of two kinds: maximal steps and reconfiguration functions.*

- *The arcs from the state $(TN(\Gamma), z)$ to the state $(TN(\Gamma), z')$ are denoted by a maximal step $u$ represented by:*

$$(TN(\Gamma), z)[u\rangle(TN(\Gamma), z')$$

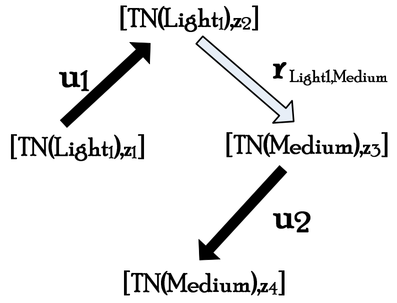


Fig 4.7 A diagram of a simplified reachability graph of FESTO MPS

*where* $\Gamma$=($P$, $T$, $F$, $W$, $CN$, $EN$, $DC$, $V$, $z_0$), $u \subseteq T$, $z = (M, D)$, $z' = (M', D')\in R(TN(\Gamma), z_0)$, $M'(P)$ = $M(P)$+$[N] \times u$, *and* $D'(p)$=$D(p) + \triangle\tau$ *if* $M(p) > 0 \wedge \forall t \in u$, $p \notin {}^\bullet t \cup t^\bullet$. *Otherwise* $D'(p) = 0$. *They are graphically represented by solid arrows.*

- *The arcs from the state* $(TN(\Gamma), z)$ *to the state* $(TN(\Gamma'), z')$ *are labeled with* $r$=($Cond$, $s$, $x$), *and are denoted in the current chapter by:*

$$(TN(\Gamma), z)[r\rangle(TN(\Gamma'), z')$$

*where* $\Gamma$=($P$, $T$, $F$, $W$, $CN$, $EN$, $DC$, $V$, $z_0$), $\Gamma'$=($P$, $T'$, $F'$, $W'$, $CN'$, $EN'$, $DC'$, $V'$, $z'$). $TN(\Gamma') = s(TN(\Gamma))$, $z'$=$x(z)$ *with* $z = (M, D) \in R(TN(\Gamma), z_0)$ *and* $z' = (M', D')$. $\forall p \in P \cap P'$, $M'(p) = M(p)$ *and* $D'(p) = D(p)$. $\forall q \in P' - P$, $M'(q) = M_0(q)$ *and* $D'(q) = 0$, *and* $M_0(q)$ *is the initial marking of* $q$ *in* $RTN$. *The reconfiguration functions are graphically represented by hollow arrows.*

**Example 6** *A simplified reachability graph of* $RTN_{FESTO}$ *is shown in Fig. 4.7. Assume that at time* $\tau_0$, *FESTO MPS is at state* $(TN(Light_1), z_1)$, *where the firing step* $u_1$ *is first enabled and fired, then the system arrives at state* $(TN(Light_1), z_2)$. *Suppose that just at this moment the system receives a reconfiguration requirement that requests the system to transform into the medium production mode. Then the reconfiguration function* $r_{L_1,M}$ *is applied at* $(TN(Light_1), z_2)$ *such that the system evolves into the state* $(TN(Medium), z_3)$, *from which the system goes on working in the medium production mode.*

R-TNCESs are different from TNCESs because of the extra reconfiguration functions. An R-TNCES can perform a group of superposed TNCESs and offer the automatic transformations of them by the addition/removal of control components and modifying the condition/event signals among them. In addition, the states coherence and temporal constraints are considered in R-TNCESs. How reconfiguration functions of an R-TNCES are implemented is illustrated in detail in the next subsection.

### 4.4.3 Reconfiguration Implementation of R-TNCESs

An R-TNCES has a set of reconfiguration functions to manage the system's dynamic reconfigurations including the structure modification and the states coherence. In this subsection, a state machine specified by a TNCES is developed to describe reconfiguration functions of an R-TNCES. In addition, a set of actuators specified by TNCESs is proposed to synchronize the state machine with the behavior module of the R-TNCES.

First of all, a state machine specified by a TNCES, which is called $Structure\_changer$, is defined, where each place corresponds to a specific TNCES of an R-TNCES. Thus, each transition corresponds to a reconfiguration function. The fact that a place $sp$ gets a token implies that the TNCES, to which $sp$ corresponds, is selected. If a transition $st$ ($\forall st \in sp^{\bullet}$) fires, then it removes the token away from $sp$ and brings it into a place $sp'$, $sp' \in st^{\bullet}$. Firing of $st$ implies that a reconfiguration function is applied. After that, the TNCES is changed into other TNCES to which $sp'$ corresponds. The $Structure\_changer$ is formalized as follows:

$$Structure\_changer{=}(P, T, F, V, m_0),$$

where $\forall t \in T$, $|{}^{\bullet}t|$ = $|t^{\bullet}|$ = 1, $\sum m_0(P) = 1$, which mean that only one place in $P$ owns a token in the initial state and only one TNCES is performed at any time, and $V : T \rightarrow \{\wedge\}$. The pre-condition of a reconfiguration function generally means a specific external instruction, or a specific system state. Therefore the pre-condition $Cond$ can be modeled by input event/condition signals from external to transitions in $Structure\_changer$.

In addition, an actuator denoted by $Actuator$ is defined for each place $sp$ in $Structure\_changer$, which is marked by $Actuator{=}Act(sp)$, such that the changed TNCES can be reactivated. Each actuator is composed of a place $mp$ and a transition $mt$ only, where ${}^{\bullet}mp{=}mp^{\bullet}{=}\{mt\}$, ${}^{\bullet}mt{=}mt^{\bullet}{=}\{mp\}$, and $M(mp) = 1$. When the place $sp$ in $Structure\_changer$ receives a token, the actuator $Actuator{=}Act(sp)$ is activated. After that, $mt$ sends event signals constantly to the corresponding control components in the TNCES to which $sp$ corresponds. In this case, the control components in the active TNCES are executable only and the others are not. An $Actuator$ is formalized as follows:

$$Actuator{=}(P, T, F, V, m_0),$$

where $|P| = |T| = 1$, ${}^\bullet mt = mt^\bullet = \{mp\}$, ${}^\bullet mp = mp^\bullet = \{mt\}$, $m_0(P) = 1$, and $V : T \rightarrow \{\vee\}$.

The set of actuators is denoted by $\sum Actuator$. Obviously, $|\sum Actuator|$ equals to the numbers of places in $Structure\_changer$. The synchronization of $Structure\_changer$ and $\sum Actuator$ is specified by event signals. Let $sp$ be a place in $Structure\_changer$ and $Actuator = Act(sp) = (mp, mt, F, W, M_0)$ be its actuator. Then $\forall st \in {}^\bullet sp$, there is a event signal from $st$ to $mt$, *i.e.*, $st \in {}^\sim mt$.

Let $\Gamma$ be the TNCES corresponding to $sp$, $\sum \mathbb{M}_\Gamma$ be the set of control components in it, $\sum \mathbb{M}_1$ be the set of shared control components used in all TNCESs of an R-TNCES $RTN$, and $\sum \mathbb{M}_2 = \sum \mathbb{M}_\Gamma - \sum \mathbb{M}_1$. Let $t \in \mathbb{M}$ ($\mathbb{M} \in \sum \mathbb{M}_2$) be the entrance transition of control component $\mathbb{M}$ by firing which $\mathbb{M}$ can be activated to work. Then there exist event signals from $mt$ to the entrance transitions in all control components in $\sum \mathbb{M}_2$, *i.e.*, $\forall \mathbb{M} \in \sum \mathbb{M}_2$, $t$ is the entrance transition of $\mathbb{M}$, $mt \in {}^\sim t$.

**Example 7** *Fig. 5.2 depicts the implementation of the R-TNCES based control model of FESTO MPS. The places $sp_1$, $sp_2$, $sp_3$, and $sp_4$ in $Structure\_changer$ correspond to $Light_1$, $Light_2$, $Medium$, and $High$, respectively. When $st_7$ fires, the reconfiguration function $r_{H,L_2}$ is implemented.*

The control components are the basic (smallest) modules in an R-TNCES. States in one control component are relatively independent with the states in other control components. Thus, by the design requirement, an equivalent TNCES model can be built for an R-TNCES following the way above, where the control module can be modeled by the synchronized $Structure\_changer$ and the set of actuators. The equivalent TNCES allows reconfigurations while the states coherence can be guaranteed during the reconfiguration processes. Given an R-TNCES, if $n1$ ($n1 \in \mathbb{N}^+$) TNCESs can be performed in its behavior module and there exist $n2$ ($n2 \in \mathbb{N}^+$) reconfiguration functions, then its control module has $2 \times n1$ places and $n1 + n2$ transitions.

In most Petri net based design methods of RDECS, a place can be defined as a state or a certain event or some special instructions such that the quantitative analysis of the net size is impractical. Compared with other formalisms, control components are applied in R-TNCESs. Therefore, it is easier for users to understand the design and facilitate the future expansion of an RDECS. Llorens and Oliver [37] implemented net rewriting systems with Petri nets. They make $n1$ ($n1 \in \mathbb{N}^+$ is the number of configurations that can be performed by the system) copies of the transitions located in different layers and the set of places in an

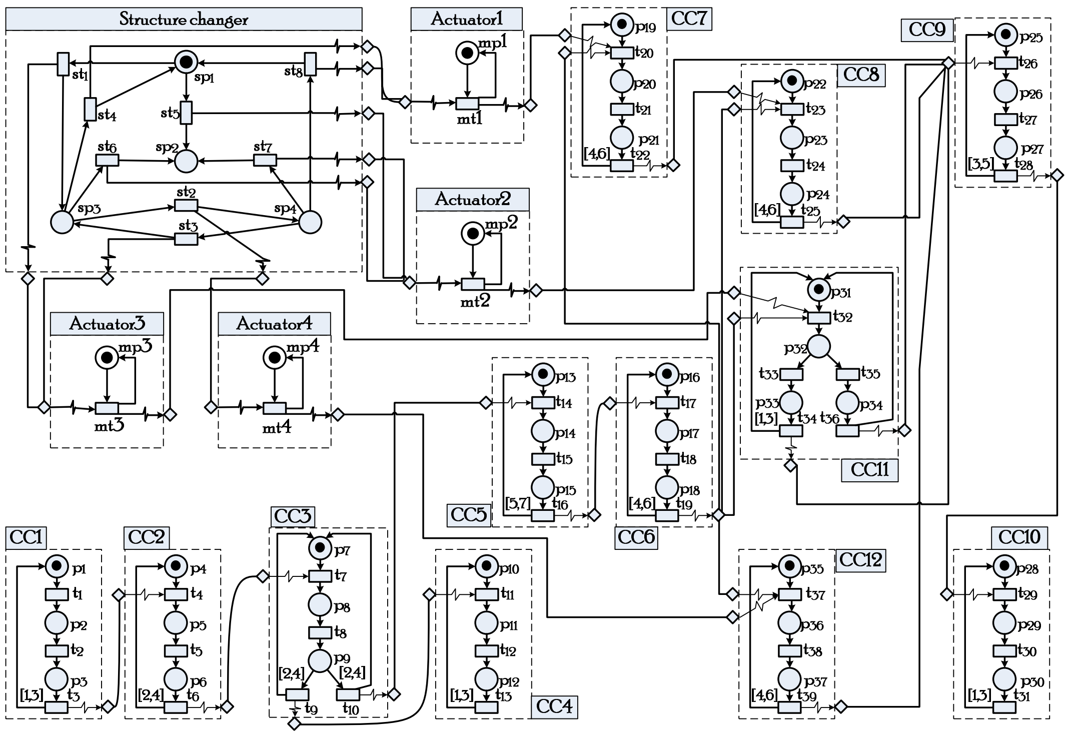


Fig 4.8 Equivalent TNCES model of FESTO's R-TNCES

exclusive layer, such that the different connections among places and transitions correspond to different system configurations. When a layer is activated, all the transitions in it and the corresponding Petri net based configuration is activated. Therefore, if net rewriting systems are applied to FESTO MPS, extra $4 \times 39 = 156$ transitions are needed for the implementation of reconfigurations. In the previous chapter, the modifications of places, transitions, and initial markings are covered only. It needs extra 13 places and 22 transitions to deal with possible reconfigurations in FESTO MPS. Whereas, in this chapter, the whole model has 8 extra places and 12 extra transitions only to cover all forms of reconfiguration scenarios including the addition/removal of control components and modifying condition/event signals among them.

## 4.5 Verification of R-TNCESs

Once an RDECS's R-TNCES model is well established, the next step is to check whether the model meets the design requirements. The checked properties include the rapid and correct response to a reconfiguration request and the valid system behavior after the application of a reconfiguration function. As far as we know, there is no work that deals with the verification of reconfigurable systems with the TNCES formalism. So far, we have not developed a specialized model-checker for R-TNCESs, thus in this chapter we use the equivalent TNCES for the verification of an R-TNCES. We note that SESA is an effective software environment for the analysis of TNCESs, which computes the set of reachable states exactly. Typical properties that can be verified are boundedness of places, liveness of transitions, and reachability of states. General functional and temporal properties can be expressed in TCL, eCTL, and TCTL, and checked manually. In this section, we provide a layer-by-layer verification method for R-TNCESs by SESA subjected to the modeling methodology presented in Section 4.3.

### 4.5.1 Verification of the Initial TNCES

Assume that $\Gamma_0$ is the default initial TNCES of an R-TNCES $RTN$ with $TN(\Gamma_0) \in \sum TN$. Other TNCESs can be obtained by implementing specific reconfiguration functions. Therefore, it is all-important to check the correctness of the initial TNCES.

The actuator of a physical process of an RDES is denoted by $act$. A chain is defined as a sequence of actuators with temporal constraints. A configuration of an RDES is described by a set of simultaneous chains and the corresponding controller is described by a set of control chains that are the sequences of control components. It is assumed that no loop exists in a chain, and no resource competition exists among the chains of a configuration. For a control chain $Control_{chain}i$, $Control_{chain}i(j)$ denotes the $j^{th}$ control component in $Control_{chain}i$, and $running(Control_{chain}i(j))$ denotes the time constraint of $Control_{chain}i(j)$.

The set of control components in a TNCES $\Gamma_i$ is denoted by $\sum \mathbb{CC}_i$. Before the verification of $\Gamma_0$, $\sum \mathbb{CC}_0$ is divided into multi-layers. The set of control chains describing $\Gamma_i$ is marked by $\sum Control_{chain}i$. If $number_{layers}i$ signifies the final number of layers of $\Gamma_i$ and $|Control_{chain}i|$ denotes the number of nodes of $Control_{chain}i$, then

$$number_{layers}i = max_{i=1}^{|\sum Control_{chain}i|}\{|Control_{chain}i|\}.$$

If $Lay_j^i$ denotes the set of control components in the $j^{th}$ layer of $\Gamma_i$, then

$$Lay_j^i = \{\cup_{i=1}^{|\sum Control_{chain}i|} Control_{chain}i(j)\}, 0 < j \leq number_{layer}i.$$

**Example 8** *$Light_1$ is the initial TNCES of $RTN_{FESTO}$. Its control system is described by the combination of $Control_{chain}1$ and $Control_{chain}2$. The control components of $Light_1$ are divided into eight layers as shown in Fig. 4.9 according to the operation order of the physical processes, where $CC_4$ and $CC_5$ are located in the fourth layer.*

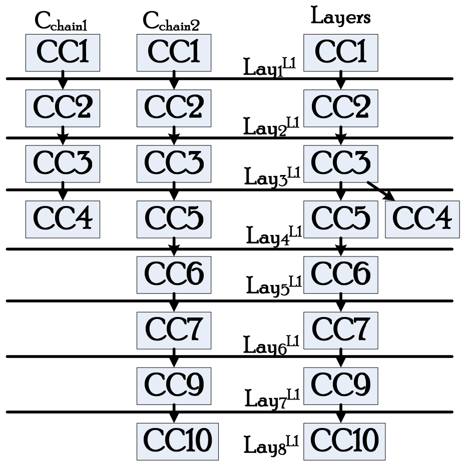


Fig 4.9 Layers of $Light_1$

The verification of $\Gamma_0$ is done layer by layer from the first one by SESA. If SESA shows that a net is bounded, live, and all the checked CTL-based properties are shown to be true, then the net is feasible with SESA. In each layer, the number of control components is not greater than the total number of control chains, *i.e.*, $1 \leq |Lay_j^i| \leq |\sum Control_{chain}i|$, and $0 < j \leq number_{layers}i$. During the verification of a layer, an abstract model denoting its external environment is constructed. The verification process of $\Gamma_0$ is described as follows:

- $Step_1$: The model-checker SESA is applied to automatically verify deadlock properties and also to manually verify CTL/eCTL-based functional and TCTL-based temporal properties (to be described by users) of the set of control components $Lay_1^0$ on the first layer. If $Lay_1^0$ is feasible with SESA, then go to the next step. Otherwise, the verification is stopped.

- $Step_i$: In each $Step_i$ $(i > 1)$, an abstract model, denoted by $Am_{i-1}^0$, is built, which used to indicate the external environment of $Lay_i^0$. The abstract model, tagged by $Am_{i-1}^0$, is constructed by the following way:

  1. Create an initial place $p_1$ with one token, an entrance transition $t_{entrance}$ from the initial place, and an output place $p_2$ from $t_{entrance}$.

  2. Build $|Lay_i^0|$ traces from the output place $p_2$. Each trace denoted by $tr_j$ has a starting transition $t_{start}^j$ coming from $p_2$ and an end transition $t_{end}^j$ that hooks up to the initial place $p_1$, where $0 < j \leq |Lay_j^0|$.

  3. Assign a time interval $[l_{i-1}^j, h_{i-1}^j]$ to the output arc $f(p_3, t_{end}^j)$ with $^\bullet t_{end}^j$=$\{p_3\}$ such that $[l_{i-1}^j, h_{i-1}^j] = \sum_{k=1}^{i-1} running(Control_{chain}j(k))$, where $0 < j \leq |Lay_j^0|$.

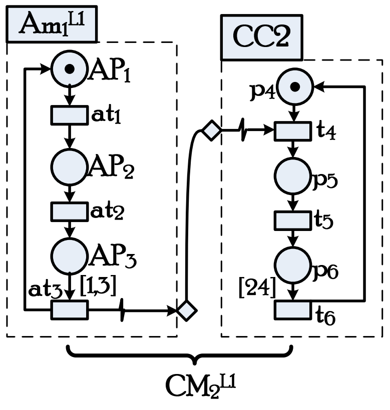


Fig 4.10 The composite module $CM_2^{L1}$

Afterwards, $Am_{i-1}^0$ is glued with $Lay_i^0$ by event signals due to the interconnection between $Lay_{i-1}^0$ and $Lay_i^0$. Then a new composite module signified by $CM_i^0$ is obtained. Finally, the model-checker SESA is applied to automatically verify deadlock properties and also to manually verify eCTL-based functional and TCTL-based temporal properties of $CM_i^0$. Specifically,

```
if $CM_i^0$ is feasible with SESA then
  if $Lay_{i+1}^0 \neq \emptyset$ then
    go to the next step verification
  else
    the verification of this TNCES is correctly done
  end if
else
  the verification is stopped
end if
```

**Example 9** *There is one control component $CC_1$ in $Lay_1^{L1}$ of $Light_1$ only. Note that $|Lay_2^{L1}|$ = 1. The abstract module $Am_1^{L1}$ and the composite module $CM_2^{L1}$ are depicted in Fig. 4.10. During the verification of $Lay_4^{L1}$, the abstract model of the external environment of ${Lay_4}^{L1}$ is first built. Note that $|{Lay_4}^{L1}| = 2$. As a result, $Am_3^{L1}$ has two traces corresponding to $Control_{chain}1$ and $Control_{chain}2$, respectively. The composite model $CM_4^{L1}$ is shown in Fig. 4.11. It is obvious that, the first three layers of $Control_{chain}1$ and $Control_{chain}2$ is the same, and a branch arises in the fourth layer. In the verification of sixth layer, $|{Lay_6}^{L1}| = 1$. Thus in the abstract model $Am_5^{L1}$, there is one trace only corresponding to $Control_{chain}2$,*

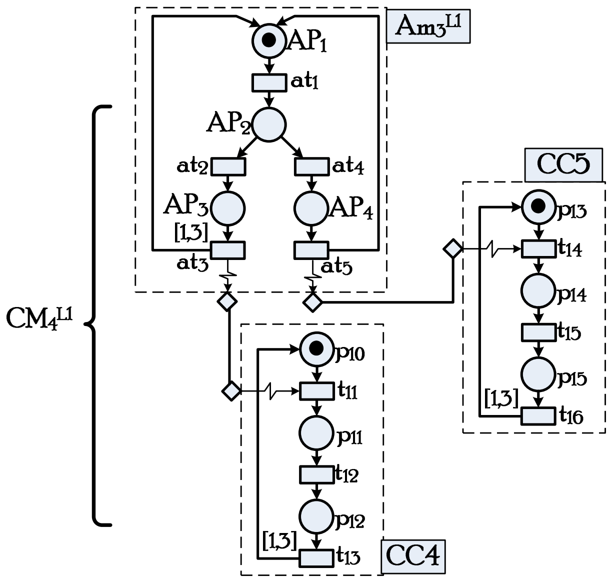


Fig 4.11 The composite module $CM_4^{L1}$

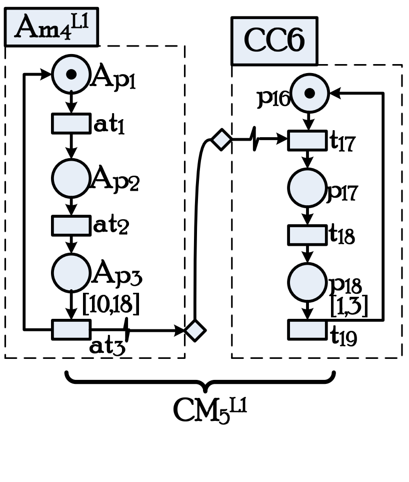


Fig 4.12 The composite module $CM_5^{L1}$

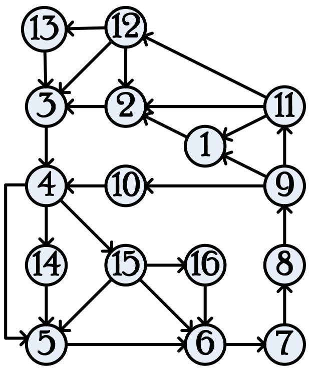


Fig 4.13 The reachability graph generated in the 4th step of $Light_1$

*as shown in Fig. 4.12.*

*In each step, eCTL/TCTL based properties specified by users are checked manually. In particular the different choices of two chains in the fourth step are checked manually. The following eCTL formula is applied to $Light_1$:*

$$z_0 \models AGAt_1XAFEt_{11}ANDt_{14}XTRUE$$

*This formula is proven to be false by SESA. Indeed, whenever $t_1$ that is the entrance transition of $Am_4^{L1}$ fires, either $CC_4$ or $CC_5$ is activated according to the test result. It is impossible that $CC_4$ and $CC_5$ are activated simultaneously. Fig. 4.13 shows the reachability graph automatically generated by SESA in the second step to check the deadlock property of the new constructed composite model $CM_2^{L1}$. From the picture, it is evident that no deadlock is in $CM_2^{L1}$.*

### 4.5.2 Verification of Other TNCESs

Other TNCESs can be obtained one by one through the implementation of certain reconfiguration functions. Let $r = (Cond, s, x)$ be a reconfiguration function, where $^\bullet r = \Gamma_a$ and $r^\bullet = \Gamma_b$. Assume that $\Gamma_a$ passes SESA successfully, then the verification of $\Gamma_b$ is illustrated as follows:

1. Divide the control components of $\Gamma_b$ into multi-layers. The resulting number of layers are denoted by $number_{layers}b$ and the set of control components in $i^{th}$ layer is denoted by ${Lay_i}^b$.

2. If, after the reconfiguration, the layers from ${Lay_k}^a$ to ${Lay_g}^a$ in $\Gamma_a$ is changed into layers from ${Lay_k}^b$ to ${Lay_{g'}}^b$ in $\Gamma_b$ while other layers remain unchanged, *i.e.*,

- ${Lay_i}^a = {Lay_i}^b, 0 < i \leq k,$
- ${Lay_{g+i}}^a$=${Lay_{g'+i}}^b$, $0 < i \leq number_{layers}a - g$, $number_{layers}a - g$=$number_{layers}b - g'$.

Then, $\Gamma'$ is verified from the $k^{th}$ layer following the layer-by-layer method proposed in the last subsection. The verification does not stop, if every layer is feasible with SESA, until the termination of the verification of the $g'^{th}$ layer.

3. Construct an abstract model $Am^b_{g'}$ for the external environment of $Lay^b_{g'+1}$. If $Am^b_{g'} = Am^a_g$, then $\Gamma'$ is correctly done. Otherwise, the remaining parts from $Lay^b_{g'+1}$ to $number_{layers}b$ should be checked again. Specifically,

```
if $CM^b_{g'}$ is feasible with SESA then
  if $Lay^b_{g'} \neq \emptyset$ then
    Construct an abstract model $Am^b_{g'}$
    if $Am^b_{g'} = Am^b_g$ then
      the verification is correctly done
    else
      go on verification
    end if
  else
    the TNCES $\Gamma_b$ is correctly done
  end if
else
  stop verification
end if
```

**Example 10** *The resulting layers of $Medium$ is shown in Fig. 4.14. The previous five layers and the last two layers of $Light_1$ and $Medium$ are the same. Therefore, the abstract model $Am^{L1}_4$ constructed in the verification process of $Light_1$ can be used in the verification of $Medium$ directly. The temporal constraints of $Am^M_6$ is $[18, 30]$, which is equal to that of $Am^{L1}_6$. As a result, ${Lay_7}^M$ and ${Lay_8}^M$ need not be checked again.*

*In particular, the TCTL property in the sixth step of $Medium$ is verified:*

$$z_0 \models EF[14, 24]p_{32} = 1$$

*This formula has been proven to be true. In fact, in the medium production mode, the drill machine $Drill_1$ or $Drill_2$ can be activated in at least $14$ time units after the system starts.*

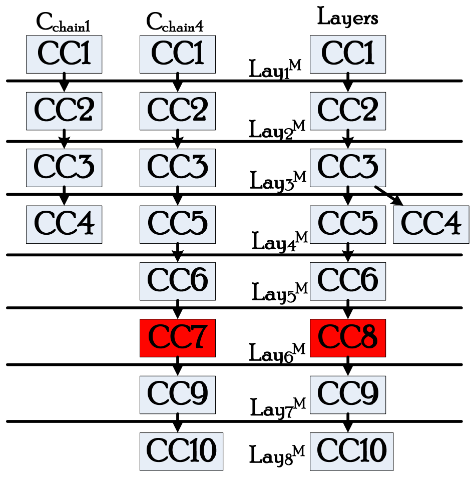


Fig 4.14 The layers of $Medium$

From the above description, the layer-by-layer method for the verification of other TNCESs is based on the fact that their original TNCESs are feasible with SESA. There exist a mass of common components between two TNCESs, which are linked by a reconfiguration function. The common parts do not need to be checked repetitively if their external environments are not changed by the reconfiguration, since they have been proven to be correct.

### 4.5.3 Verification of the Control Module

An R-TNCES can be implemented by an equivalent TNCES as shown in Section 4.4.3. In this subsection, we focus on the cases when a reconfiguration request or an error occurs, whether the control module can respond and select a proper TNCES. The following CTL formula is proposed to the control module of $RTN_{FESTO}$ in Fig. 5.2:

$$z_0 \models AGAst_5XAFEmt_1Xsp_1$$

This formula has been proven to be false by SESA. In fact, when $st_5$ fires, $Light_2$ is selected, and the $Actuator_2$ is activated rather than $Actuator_1$. Thus, $mt_1$ in $Actuator_1$ cannot get enabled under this circumstance. The following formula has been proven to be true when $Light_1$ is selected.

$$z_0 \models AGAst_4ORst_8XAFEmt_1Xsp_1$$

The complexity of the layer-by-layer method for the verification of the TNCESs of an R-TNCES is considered. Suppose that $n_{complexity}$ is the upper bound of the number of resulting layers of a TNCES, and it also denotes the total steps for the construction of the composite model in each layer. Therefore, the complexity of the automatic layer-by-layer construction problem of a TNCES is $O(n_{complexity}^2)$. Compared with the refinement-based verification method of plant control systems in [74], we have following distinctions and improvements: 1) Control components are applied directly to model the control system in this chapter, whereas in [74], a place in the plant model is refined to the corresponding controller specified by a control component, where reconfigurations are not allowed. 2) The control chains describing the behavior of a TNCES are merged together before the separation of the control components into multi-layers. Therefore, the checked net is minimized in each layer, whereas the chains are checked simultaneously in [74] without the combination. 3) At each step, we build an abstract model as the external environment of the underlying layer. Whereas in [74], an abstract model is built at each step as a rough model of the system, in which a place is refined according to certain refinement rules of TNCESs, by which the net's properties are preserved. 4) The similarity among various TNCESs of an R-TNCES is considered, such that the comprehensive verification of the TNCESs can be simplified, except the initial one.

### 4.5.4 System Correctness

The correctness of the proposed layer-by-layer verification approach is proven in this section. Let $number_{layers}i$ be the sum of layers of a TNCES $\Gamma_i$ and $\sum Control_{chains}i$ be the set of control chains describing $\Gamma_i$.

**Theorem 1** *If the generated composite model of the initial TNCES $\Gamma_0$ in each step is feasible with SESA, then $\Gamma_0$ is correct according to user requirements.*

***Proof***: During the verification of the $\Gamma_0$, a sequence of composite modules is generated, which is denoted by $CM_1^0 CM_2^0 ... CM_{number_{layers}i}^0$. Suppose that the generated composite model in each step is feasible with SESA, but certain functional or temporal properties are not satisfied. We have two cases:

- Case1. Let $bound_{chain}i$ denote the temporal constraint of a control chain $Control_{chain}i$, with $bound\ _{chain}i = \sum_{k=1}^{|Control_{chain}i|} running\ (Control\ _{chain}i(k))$ and $0 < i \leq |\sum Control_{chains}|$. If a temporal property $\phi$ is not satisfied by the whole TNCES, there exists a composite module $CM_{|Control_{chain}i|}^0$, generated in the $|Control_{chain}i|^{th}$ step of the verification ($0 < |Control_{chain}i| \leq number_{layers}$), which does not meet the corresponding temporal bound $bound_{chain}i$. In this case, we have

$$h^i_{|Control_{chain}i|} > bound_{chain}i,$$

where $h^i_{|Control_{chain}i|}$ is the upper bound of the time interval of the trace corresponding to $Control_{chain}i$ in $CM^0_{|Control_{chain}i|}$. That is to say, the upper bound $h^i_{|Control_{chain}i|}$ of the composite module generated in the $|Control_{chain}i|^{th}$ step is larger than the required temporal constraints. Nevertheless, in the proposed approach, if each composite module satisfies with the time constraints, then

$$h^i_{|Control_{chain}i|} \leq bound_{chain}i.$$

- Case2. Let $\varphi$ be a functional property that is not satisfied by $\Gamma_0$. According to the assumption, a subset of composite models, to be generated in different layers, does not satisfy $\varphi$. This is incompatible with the assumption.

Thus the theorem is true. ■

**Theorem 2** *Given two TNCESs $\Gamma_a$ and $\Gamma_b$ of an R-TNCES, $\Gamma_a$ is proved to be correct by SESA. Suppose that only the layers from ${Lay_k}^a$ to ${Lay_g}^a$ in $\Gamma_a$ are different from that from ${Lay_k}^b$ to ${Lay_{g'}}^b$ in $\Gamma_b$. If the external environment of the un-changed parts of $\Gamma_a$ is not changed, the repetitive verification of them can be avoided during the verification process of $\Gamma_b$.*

***Proof***: Obviously, the first $k$ layers of $\Gamma_b$ need not to be checked repetitively due to Theorem1. From the $(g' + 1)^{th}$ step of the verification process, whether the verification can be avoided depends on the external environment of $Lay^b_{g'+1}$, where $Lay^b_{g'+i} = Lay^a_{g+i}$ with $0 < i \leq |\sum Control_{chains}b| - g'$. If the constructed abstract model in the $(g+1)^{th}$ step of $\Gamma_a$ is the same with the one generated in the $(g'+1)^{th}$ step of $\Gamma_b$, *i.e.*, $Am^a_g = Am^b_{g'}$, the generated composite module $CM^b_{g'+1} = CM^a_{g+1}$. Therefore, the latter generated sequence of composite modules $CM^b_{g'+1}CM^b_{g'+1}...CM^b_{|\sum Control_{chains}b|}$ equal $CM^a_{g+1}CM^b_{g+1}...CM^a_{|\sum Control_{chains}a|}$. Therefore, the residual layers need not be checked repetitively. However, if $Am^a_g \neq Am^b_{g'}$, the latter generated sequence of composite modules $CM^b_{g'+1}CM^b_{g'+1}...CM^b_{|\sum Control_{chains}b|}$ is changed. Thus the residual layers has to be verified one by one again, although the layers are maintained unchanged. From the above statements, the theorem is true. ■

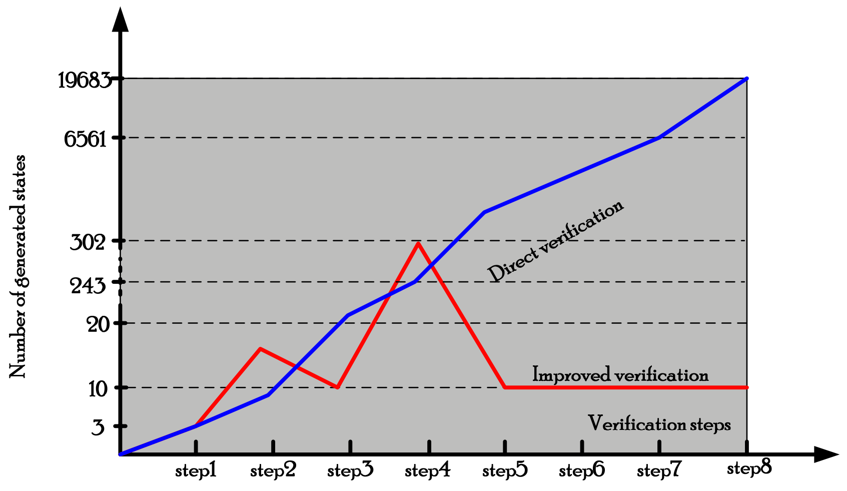


Fig 4.15 Improved verification of $Light_1$

### 4.5.5 Discussion

In the benchmark production system, $Light_1$ is the initial production mode. Eight reachability graphs corresponding to the eight steps of the layer-by-layer verification process are computed by SESA. Fig. 4.15 demonstrates how the developed verification method is improved comparing with the traditional verification method for NCESs. The number of generated states by the proposed approach for $Light_1$ is $371$, whereas it is $19,683$ if a verification algorithm without any improvement is directly applied. Table 4.2 depicts the generated state number during the verification of $Medium$. The previous five and the last two layers of $Medium$ are not checked, because the external environment of $Lay_7^{L1}$ and $Lay_8^{L1}$ is not changed by implementing $r_{L1,M}$ .

| Steps | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| States number of $Light_1$ | 3 | 16 | 10 | 302 | 10 | 10 | 10 | 10 |
| States number of $Medium$ | 0 | 0 | 0 | 0 | 0 | 13 | 0 | 0 |

Table 4.2 State number in the verification of $Medium$

Fig. 4.16 reveals the result of an experimental study to compute the number of generated states for the automatic construction of feasible 100 steps to verify a TNCES. We assume that there are 100 layers in a TNCES and the behavior of this TNCES is described by three parallel chains with the same length. Assume that each layer generates 1000 states. It is possible to generate $1000^{100} = 1.e + 300$ states if SESA is applied directly to this TNCES. However, if our method is applied, during the verification, 100 abstract models are built. Each abstract model has three traces (five places). Thus in each step it gen-

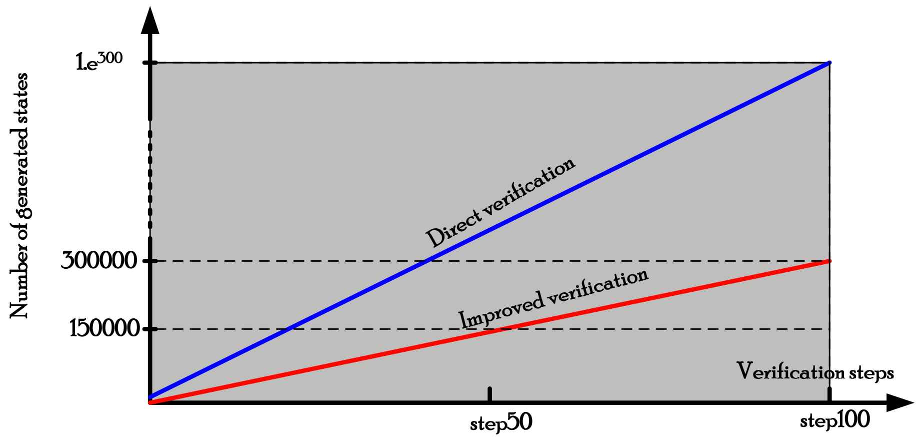


Fig 4.16 Improved verification of 100 steps

erates $5 * 1000 = 5000$ states. Therefore, during the verification process, it generates $5000 * 100 = 500,000$ states only.

## 4.6 Summary

This chapter introduces R-TNCES, a new formalism for modeling and verification of RDECSs. Compared with the previous studies on formal methods for RDECS, the functional and temporal specifications are optimized, and more forms of reconfiguration scenarios are covered such as the addition/removal of control components and the modifications of condition/event signals among them. An R-TNCES architecture is composed of a behavior module and a control module. The former is a union of a group of TNCESs that are composed of control components with uniform interfaces. The latter handles the automatic reconfigurations of these TNCESs. A reconfiguration function has not only a structure modification instruction to dispose the structure reconfiguration but also a state correlation function to assure the coherence of states before and after any implementation of reconfiguration scenarios. Therefore, an R-TNCES is such a formalism that guarantees the correctness of an RDECS from the viewpoint of the model.

A layer-by-layer verification method for R-TNCESs is also proposed by using the model-checker SESA. The initial TNCES of an R-TNCES is checked first. Its control components are divided into multi-layers in terms of predetermined execution orders and then checked layer by layer. An abstract model denoting the external environment of the underlying layer is constructed during the process, and the obtained composite net is checked by the model-checker SESA. Meanwhile, the functional and temporal properties specified by users are checked manually. Other TNCESs are obtained one by one by implementing certain reconfiguration functions. The similarity among the TNCESs is considered, which helps to simplify the verification process. It is proved that if the external environment of the unchanged parts is not changed by reconfigurations, the repetitive verification of the them can be avoided. This solution controls the complexity of model-checking of an R-TNCES. The benchmark production system FESTO MPS is taken as a whole running example in the context of this chapter. It shows that the R-TNCES is a convenient formalism for modeling and analyzing RDECSs.

# Chapter 5 Coordination of R-TNCESs

A distributed reconfigurable discrete event control system (DRDECS) is composed of several networked reconfigurable subsystems. In order to realize the system functions, these reconfigurable subsystems communicate and coordinate with each other. Any automatic reconfiguration applied to a subsystem may cause risk to others and even to the safety of the whole system. This chapter reports a new coordination method for a DRDECS, where each subsystem is modeled by an R-TNCES. A virtual coordinator together with a communication protocol between it and subsystems is developed in order to achieve two aims: 1) to coordinate subsystems with an optimal coordination solution by using *Judgement Matrices* while multiple subsystems require global reconfigurations and 2) to reduce the exchanged messages between the coordinator and these subsystems. Furthermore, for the purpose of checking the functional and temporal properties of a DRDECS with a virtual coordinator, the CTL based model checking method is applied. Finally, a hypothetic manufacturing plant is used as a running example to illustrate the work.

## 5.1 Motivation

A distributed discrete event control system (DDECS) is a discrete-state, event-driven system whose evolution depends entirely on the occurrence of asynchronous discrete events over time [22]. A DDECS allowing dynamic reconfigurations is called in this chapter a distributed reconfigurable discrete event control system (DRDECS). A dynamic reconfiguration means the automatic change of a running system while the system performance is not reduced [83]. It is always caused by continuous changes of the execution environment or faults detecting of the system components. A great deal of real-world systems can be represented as DRDECSs such as traffic control systems, automobile electronic systems, and manufacturing systems [41], [43], [84], [85], [86].

Physically, a DRDECS is composed of a set of networked reconfigurable discrete event control subsystems. Possible reconfiguration scenarios of a DRDECS can be divided into two types: local reconfigurations and global reconfigurations. The former is implemented by subsystems independently without noticing others, which has been studied in many works [81], [1], [87], [75], [44], [88]. The latter involves parts or all subsystems. Therefore, the coordination of subsystems in such a case is of great importance and significance in many cases, where the communication among them plays a key role such that all the subsystems work effectively in a group setting [75], [53], [54], [89]. This chapter focuses on global reconfigurations of a DRDECS.

As discussed in Chapter 1, several research works have been done in recent years, which define inter-agent communication protocols for the coordination of various components [49], [50], [51], [52], [53], [54], [55]. All those existing communication modes are effective in their application fields. However, the rate of exchanged messages is an important criterion in order to guarantee an acceptable level of satisfaction and robustness [50], [51]. A multi-agent architecture is proposed in [56] to deal with the coherence of distributed devices, where the exchanged messages among agents are reduced significantly compared with a direct point-to-point communication mode among agents. The method in [56] gives a coordination solution for multiple concurrent requirements. However, the solution only aims to satisfy the requirement with the highest priority. An optimal coordination solution for all concurrent reconfiguration requirements is not studied.

In order to better cope with the coordination problems of a DRDECS, this chapter develops a novel virtual coordinator together with a communication protocol that defines interaction rules between subsystems and this coordinator. The proposed method is competent to treat all concurrent reconfiguration requirements with an optimal coordination solution that is selected by using *Judgement Matrices* handled by the coordinator, while the exchanged messages are well-controlled.

In this chapter, the virtual coordinator and the communication protocol are modeled by TNCESs. Reconfigurable subsystems of a DRDECS are modeled by reconfigurable timed net condition/event systems (R-TNCESs) [87]. After the generation of models of a DRDECS, the model checker SESA is applied to check whether they fulfill design requirements. All required or forbidden properties are specified by CTL as well as its extensions. The alternative of using a liner time logic is ruled out because any model checker for such a logic must have high complexity [90]. Finally, the communication protocol is evaluated in this study by computing the exchanged messages among subsystems and the coordinator.

## 5.2 Reconfigurable Coordination of a DRDECS

A distributed reconfigurable discrete event control system (DRDECS) is composed of a group of networked reconfigurable subsystems. The possible reconfiguration scenarios in a DRDECS are divided into two categories: global and local reconfigurations. Each subsystem is able to handle its own local reconfigurations independently without asking

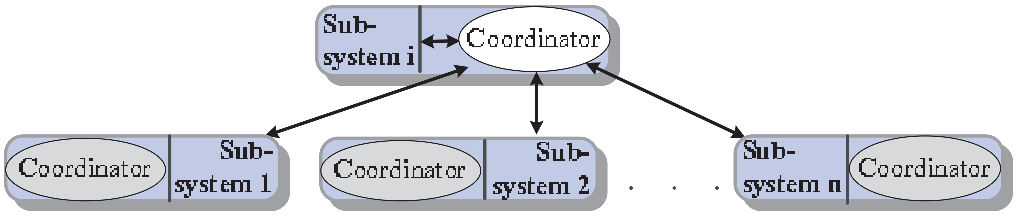


Fig 5.1 Architecture of a DRDECS

for a permission from other subsystems. However, if a subsystem wants to apply a global reconfiguration that may cause turbulence to other running subsystems, it has to apply for a permission from them. In order to achieve the targets of 1) controlling the amount of exchanged messages between subsystems and 2) dealing with all concurrent requirements for global reconfigurations with an optimal coordination solution, a novel virtual coordinator handling *Judgement Matrices* together with a communication protocol is developed.

### 5.2.1 Specification of a DRDECS

Assume that a DRDECS $\mathscr{S}$ is composed of $\mathbb{n}$ networked reconfigurable subsystems. Then a DRDECS $\mathscr{S}$, as shown in Fig. 5.1, is defined by

$$\mathscr{S} = (\Sigma, \varpi, \xi),$$

where $\Sigma$ is a set of $\mathbb{n}$ subsystems, $\varpi$ is a virtual coordinator handling a set of *Judgement Matrices*, and $\xi$ is a communication protocol that defines the interaction rules between $\Sigma$ and $\varpi$.

The virtual coordinator exchanges messages with subsystems respectively in such a DRDECS rather than subsystems communicate directly. Whenever a subsystem desires to apply a global reconfiguration, it should send a request to the coordinator first to get a permission. A coordination solution, *i.e.*, how the whole system should react according to the received reconfiguration requirements, is made in the virtual coordinator by using *Judgement Matrices* before commands are sent to relevant subsystems. Any subsystem cannot apply a global reconfiguration until it receives a command from the virtual coordinator. This chapter assumes that the virtual coordinator runs on one of the subsystems as shown in Fig. 5.1. If its host breaks down or is deactivated in a particular condition, a coordinator equal to the original one in other subsystems is selected and activated to be used. Therefore, the coordinator is said to be *virtual*.

A subsystem, to be denoted by $\varepsilon$, in a DRDECS is a reconfigurable discrete event control system, which can be presented as follows:

$$\varepsilon = (\bigcup(Cof), Ref, Cof_0).$$

$\bigcup(Cof)$ is a set of all possible configurations of $\varepsilon$. $Ref$ is a set of all global reconfiguration scenarios in $\varepsilon$. $Cof_0$ is the initial configuration, from which $\varepsilon$ starts. The configurations in $\bigcup(Cof)$ are in a 'family', which means the configurations are similar in aspects of input types, main functions and output types. A reconfiguration scenario general means to modify the system structure by adding/removing hardware/software components, justifying parameters, or changing logic relationship between components.

Suppose that a subsystem $\varepsilon_i$ ($\varepsilon_i \in \Sigma$) has $n_i$ configurations and $m_i$ global reconfiguration scenarios. A *distributed configuration* of a DRDECS, denoted by $D_\Gamma$, is defined by an $\mathbb{n}$-tuple:

$$D_\Gamma = (\varepsilon_{1,k_1}, \varepsilon_{2,k_2}, ..., \varepsilon_{\mathbb{n},k_\mathbb{n}}),$$

where $\varepsilon_{i,k_i}$ identifies the ${k_i}^{th}$ configuration ($i \in \{1, 2, ..., \mathbb{n}\}$, $k_i{\in}\{1, 2, ..., n_i\}$) of $\varepsilon_i$. The set of the distributed configurations of a DRDECS is denoted by $\mathscr{D}$. The set of possible distributed reconfiguration scenarios, denoted by $\mathscr{R}$, is a mapping:

$$\mathscr{R} : \mathscr{D} \to \mathscr{D}.$$

A distributed reconfiguration scenario, denoted by $D_r$, is defined by an $\mathbb{n}$-tuple:

$$D_r = (r_{1,h_1}, r_{2,h_2}, ..., r_{\mathbb{n},h_\mathbb{n}}),$$

where $r_{i,h_i}$ ($i \in \{1, 2, ..., \mathbb{n}\}$ and $h_i \in \{1, 2, ..., m_i\}$) is the ${h_i}^{th}$ reconfiguration scenario of $\varepsilon_i$. If $r_{i,h_i}$ exists (e.g. true), the global reconfiguration scenario $r_{i,h_i}$ is implemented in $\varepsilon_i$ during the implementation of this distributed reconfiguration scenario $D_r$. Otherwise (e.g. false), no global reconfiguration scenario is executed in $\varepsilon_i$.

### 5.2.2 Reconfigurable Coordination of a DRDECS

Let $D'_\Gamma = (c_{1,k_1}, c_{2,k_2}, ...., c_{\mathbb{n},k_\mathbb{n}})$ be the on-going distributed configuration. Suppose that subsystem $\varepsilon_i$ desires to reconfigure itself from the ${k_i}^{th}$ configuration into the $g_i^{th}$ configuration ($k_i, g_i \in \{1, 2, ..., m_i\}$ with $k_i \neq g_i$), then it first sends the following requirement to the coordinator $\varpi$:

$$Request(From\_\varepsilon_i, To\_\varpi, Target\_g_i, Cause).$$

$From\_\varepsilon_i$ indicates that the requirement is sent from subsystem $\varepsilon_i$. $To\_\varpi$ denotes that the requirement is sent to the coordinator $\varpi$. $Target\_g_i$ means that the sender, *i.e.*, $\varepsilon_i$, would like to reconfigure itself from its current configuration into its ${g_i}^{th}$ configuration, which is called its *target configuration* for sending this requirement. $Cause$ denotes the reason of this requirement.

Let $\beta_{req}$ denote the number of concurrent reconfiguration requirements received by the coordinator. They are sent from different subsystems respectively. After that, an optimal coordination solution for these concurrent requirements is computed by the coordinator. We use $D''_{\Gamma} = (o_1, o_2, ..., o_{\mathbb{n}})$ to denote the final selected optimal coordination solution for the received $\beta_{req}$ requirements, where $o_i \in \bigcup(Cof)_i$, $i \in \{1, ..., \mathbb{n}\}$. The optimal coordination solution has the highest *satisfaction ratio* to the received requirements and the lowest *change ratio* to other subsystems that do not desire reconfigurations. The *satisfaction ratio* can be defined by $S_r = N_s/\beta_{req}$, where $N_s$ is the number of subsystems that send requirements and receive permission for them. The *change ratio* is defined by $C_r = N_c/(\mathbb{n} - \beta_{req})$, where $N_c$ is the number of subsystems that do not send requirements but have to apply reconfigurations according to the coordination solution.

During a coordination process, *Judgement Matrices* handled by the coordinator play a key role, which are pre-defined by designers and define whether a distributed configuration is legal, *i.e.*, admissible, when a subsystem is in a particular configuration. A *Judgement Matrix* corresponds to a possible reconfiguration requirement according to its target configuration, into which the sender desires to transform.

Let $\mathscr{M}$ be such a matrix corresponding to the reconfiguration requirement $Request$ $(From\_\varepsilon_i, To\_\varpi, Target\_g_i, Cause)$. It is an $\mathbb{m} \times \mathbb{n}$ matrix, where $\mathbb{m}$ is the number of possible distributed configurations while the subsystem $\varepsilon_i$ is in the ${g_i}^{th}$ configuration and $\mathbb{n}$ is the number of subsystems. In the $i^{th}$ column of this matrix, $\forall a, b \in \{1, 2, ..., \mathbb{m}\}$ ($a \neq b$), we have $\mathscr{M}[a, i] = \mathscr{M}[b, i]$. Obviously, a row vector of such a matrix corresponds to a particular distributed configuration. For computing the optimal coordination solution for the received $\beta_{req}$ reconfiguration requirements, only their corresponding *Judgement Matrices* are needed. The set of needed *Judgement Matrices* is denoted by $\Phi$.

Before computing the optimal coordination solution, the concurrent reconfiguration requirements received by the coordinator are divided into two levels: critical and common requirements according to their causes. Accordingly, the subsets of *Judgement Matrices* of critical and common requirements are denoted by $\Phi_{cr}$ and $\Phi_{co}$, respectively. Generally, critical ones are those caused by faults detecting or significant changes in environment.

Whereas common ones are caused by some optional behavior such as energy saving. The details how an optimal coordination solution is obtained by using *Judgement Matrices* are shown below.

```
Input:$D'_\Gamma$, $\Phi_{cr}$, and $\Phi_{co}$
Output:$D''_\Gamma$
  for All matrices in $\Phi_{cr}$, check if there exists a row, in which all subsystems that send
  critical requirements are in their target configurations;
  Exists?
    Y: Save it into $PP_1$;
  end for
  if $PP_1 = \emptyset$;
    then End, Output: No coordination solution;
  else if
    for Each row in $PP_1$, check if there is a row, in which all subsystems that send common
  requirements are in their target configuration;
    Exists?
      Y: Save it into $PP_2$;
    end for
    if $PP_2 \neq \emptyset$;
      then Select a row in $PP_2$ as $D''_\Gamma$, which has the lowest change ratio;
    else if
  Select a row in $PP_1$ as $D''_\Gamma$, which has the highest satisfaction ratio and the lowest change
  ratio;
  end if
```

The possible reactions of a subsystem $\varepsilon_i$ according to the coordination solution are shown in TABLE 5.1. The first column denotes whether or not $\varepsilon_i$ sends a reconfiguration requirement to the coordinator. If it sends a requirement, it is marked by its target configuration $g_i$, else $-$. The second shows the possible coordination solutions for the received concurrent requirements by the coordinator. $Accept$ (*resp.*, $Reject$) means its requirement is accepted (*resp.*, rejected), $Reconfigure$ (*resp.*, $-$) denotes the subsystem should (*resp.*, should not) apply a local reconfiguration, and $Reject\&Reconfigure$ means its requirement is rejected but it also has to apply a reconfiguration that transforms it to a non-target configuration. The third column denotes the correct reconfiguration scenario that the subsystem should apply, where $c_{i,k_i}$ denotes the original configuration before this global reconfiguration, $o_i$ denotes its configuration after the reconfiguration, and $-$ means no reconfiguration is applied.

Table 5.1 Possible reactions of a subsystem

| Target | Solution | Reaction |
|---|---|---|
| $g_i$ | *Accept* | $c_{i,k_i} \rightarrow o_i$ $(g_i = o_i)$ |
| $g_i$ | *Reject* | – |
| $g_i$ | *Reject* & *Reconfigure* | $c_{i,k_i} \rightarrow o_i$ $(g_i \neq o_i)$ |
| – | – | – |
| – | *Reconfigure* | $c_{i,k_i} \rightarrow o_i$ |

If the coordination solution is $Accept$, $Reject$ & $Reconfigure$, or $Reconfigure$, ***i.e.***, $\forall i \in \{1, 2, ..., \mathbb{m}\}$, $c_{i,k_i} \neq o_i$, then the coordinator sends the following signal to the subsystem $\varepsilon_i$ to command it to transform from the ${k_i}^{th}$ configuration to the ${o_i}^{th}$ configuration:

$$Order(From\_\varpi, To\_\varepsilon_i, Target\_o_i).$$

If a subsystem $\varepsilon_i$ receives such a signal from the coordinator, a global reconfiguration scenario is implemented, which transforms $\varepsilon_i$ from the $k_i^{th}$ configuration to the $o_i^{th}$ configuration. If the coordination solution is $Reject$, then the coordinator sends the following signal to $\varepsilon_i$ to reject its requirement:

$$Reject(From\_\varpi, To\_\varepsilon_i, Target\_g_i).$$

In a word, a subsystem expecting a global reconfiguration scenario should first send a request to the coordinator. In order to obtain an optimal coordination solution for the received multiple reconfiguration requirements, *Judgement Matrices* are applied.

## 5.3 Modeling of DRDECSs

It is essential to build mathematical models for a new method in order to perform quantitative and qualitative analysis. TNCESs are an advisable choice to model distributed discrete event control systems, which have been applied in many works. The R-TNCES formalism is an extensions of TNCESs, which allows dynamic reconfigurations such as adding/removing places, transitions, arcs, signals, modules and updating states. R-TNCESs inherit all symbols and graphical representations from TNCESs. In the following, we introduce a hypothetic DRDECS before the modeling.

### 5.3.1 Benchmark Production System

A hypothetic DRDECS $Sys$ composed of two real-physical reconfigurable manufacturing subsystems: $\varepsilon_F$ and $\varepsilon_E$,[1] is applied to illustrate the proposed method. It is assumed that the two subsystems collaborate to manufacture workpieces and some particular reconfiguration scenarios can be applied to them according to well-defined conditions.

In $\varepsilon_F$, a new incoming workpiece is tested in color, height, and material before it is drilled. If the drilled workpiece is qualified, it is transformed onto $\varepsilon_E$ directly. In $\varepsilon_E$, workpieces are put in tins and delivered into different storage units according to their dimensions through a conveyor. We assume that $\varepsilon_F$ has two drillers $Dr_1$ and $Dr_2$. It has three behavior modes: light, medium, and high according to different manufacturing speeds. In the light mode only one driller is used. In the default light mode $L_1$, only $Dr_1$ is used. The light mode $L_2$ is a substitution of $L_1$ when $Dr_1$ breaks down. In medium mode $M$, $Dr_1$ and $Dr_2$ are used alternatively. In high mode $H$, $Dr_1$ and $Dr_2$ are used simultaneously. Accordingly, $\varepsilon_E$ has two policies $P_1$ and $P_3$ according to different product types. We assume $\varepsilon_E$ has two jack stations $J_1$ and $J_2$. In $P_1$, workpieces are placed by $J_1$ or $J_2$ in a tin before closing the tin with a cap. After that, the produced workpiece is removed into the first storage unit. In $P_3$, both $J_1$ and $J_2$ are used to place workpieces from $\varepsilon_F$ into a tin, at this time a tin has two workpieces jacked by $J_1$ and $J_2$ separately before the tin is closed and displaced thereafter on the belt to the second storage unit. The mode $P_2$ is a substitution of $P_1$ when $J_1$ breaks down, in which $J_2$ is used only.

The possible reconfiguration scenarios in $\varepsilon_F$ are $L_1 \rightarrow L_2$, $L_1 \rightarrow M$, $M \rightarrow L_1$, $M \rightarrow L_2$, $M \rightarrow H$, $H \rightarrow L_1$, $H \rightarrow L_2$, $H \rightarrow M$. The possible reconfiguration scenarios in $\varepsilon_E$ are $P_1 \rightarrow P_2$, $P_1 \rightarrow P_3$, $P_3 \rightarrow P_1$, $P_3 \rightarrow P_2$. We assume that all of them are global reconfiguration scenarios.

### 5.3.2 Formal Models

As described in the previous chapter, a reconfigurable discrete event control system can be easily modeled by an R-TNCES. Therefore, a reconfigurable subsystem in a DRDECS in this chapter can be represented by

$$\varepsilon = (\Gamma_0, \mathcal{B}, \mathcal{R}),$$

where $\Gamma_0$ is a TNCES modeling the initial configuration. $\mathcal{B}$ is the set of TNCESs corresponding to all possible configurations of $\varepsilon$. $\mathcal{R} = \{\mathcal{R}_L, \mathcal{R}_G\}$ is a set of reconfiguration

[1] Their prototypes are FESTO and EnAS as described in Chapter 3 and Chapter 4, which are available in Martin Luther University: *http://aut.informatik.uni-halle.de*.

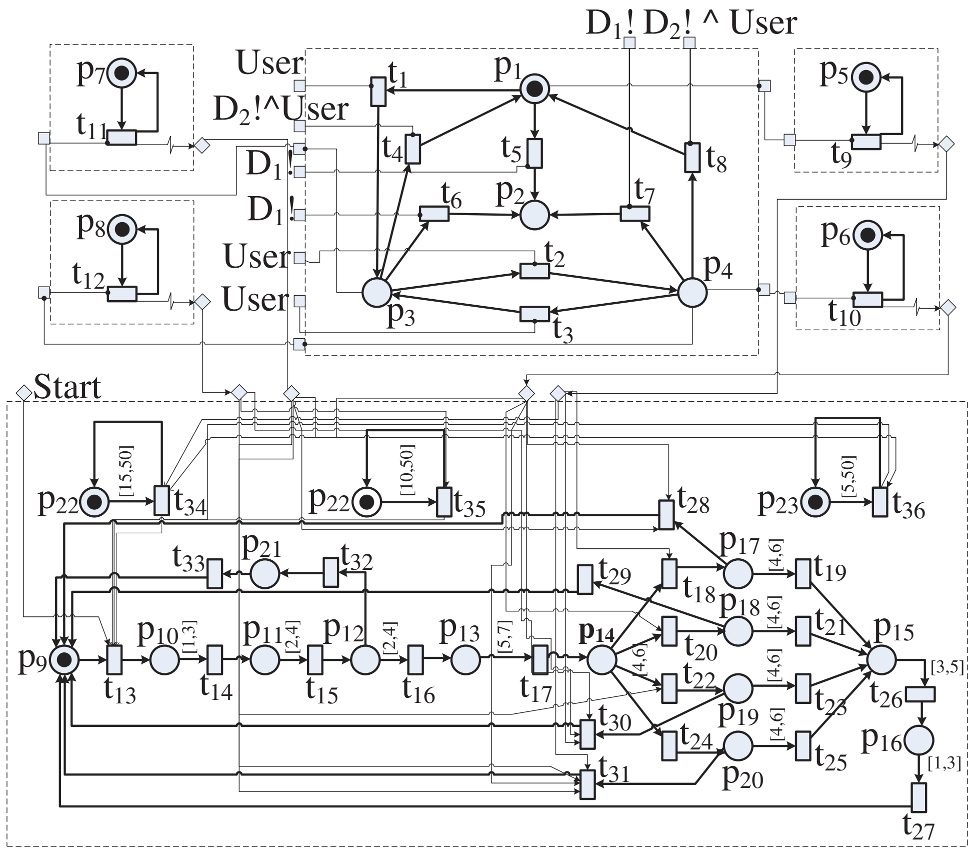


Fig 5.2 R-TNCES model of $\varepsilon_F$

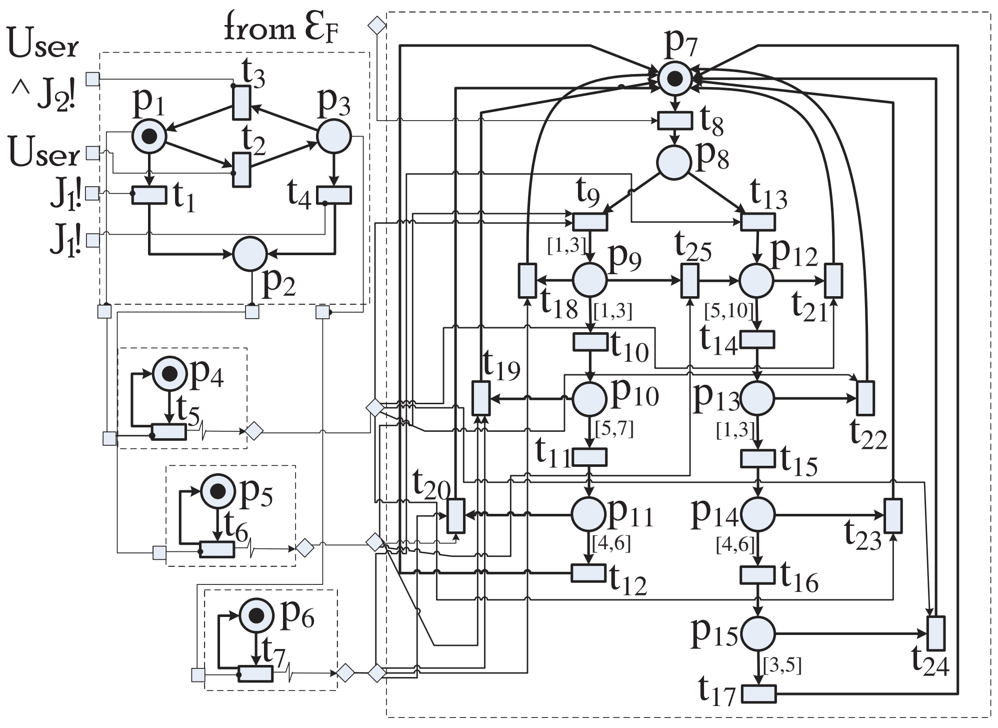


Fig 5.3 R-TNCES model of $\varepsilon_E$

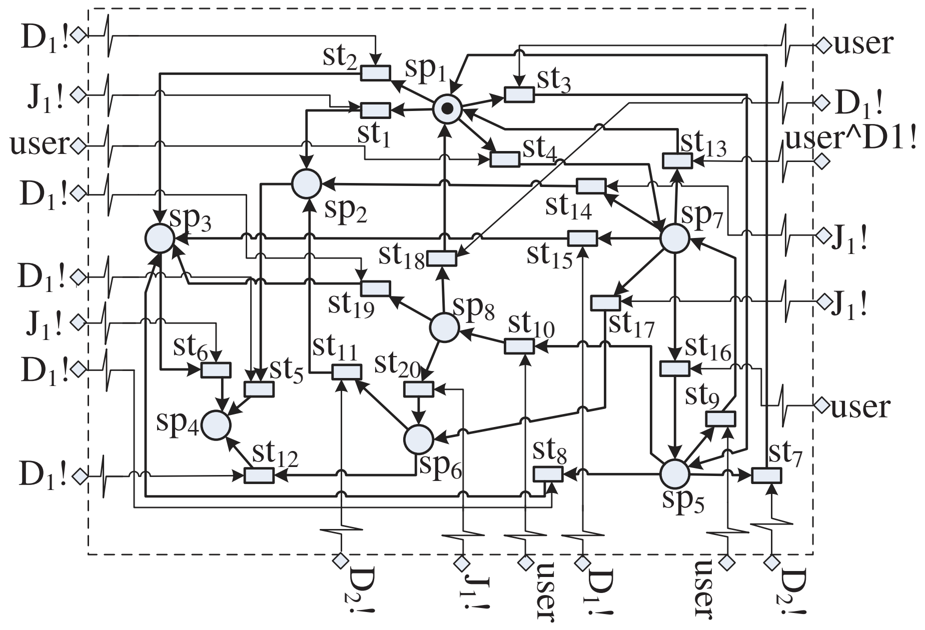


Fig 5.4 Coordinator of $\varepsilon_F$ and $\varepsilon_E$

functions, where $R_L$ models the set of local reconfiguration scenarios and $R_G$ models the set of global reconfiguration scenarios with $R_L \cap R_G = \emptyset$. The local reconfiguration function can be implemented without input condition or input signals. Whereas a global reconfiguration function needs to be triggered by some input events or some outside condition before it is implemented. The R-TNCES models of $\varepsilon_F$ and $\varepsilon_E$ are shown in Fig. 5.2 and Fig. 5.3, respectively.

The coordinator $\varpi$ records the on-going distributed configuration. The implemented distributed reconfiguration scenario decides the particular distributed configuration into which the whole system can transform. A TNCES-based state machine is applied to model the behavior of a coordinator, where each place corresponds to a specific distributed configuration of the DRDECS and the firing of a transition means the implementation of a particular distributed reconfiguration scenario from a distributed configuration into another. The TNCES-based model of the coordinator of $\varepsilon_F$ and $\varepsilon_E$ is shown in Fig. 5.4.

**Example 11** *In $Sys$, the set of* Judgement Matrices *are shown below:*

$$\mathscr{M}_{1,1}=\begin{pmatrix} L_1 & P_1 \\ L_1 & P_2 \end{pmatrix} \mathscr{M}_{1,2}=\begin{pmatrix} L_2 & P_1 \\ L_2 & P_2 \end{pmatrix} \mathscr{M}_{1,3}=\begin{pmatrix} M & P_1 \\ M & P_2 \end{pmatrix}$$

$$\mathscr{M}_{1,4}=\begin{pmatrix} H & P_3 \end{pmatrix} \mathscr{M}_{2,1}=\begin{pmatrix} L_1 & P_1 \\ L_2 & P_1 \\ M & P_1 \end{pmatrix} \mathscr{M}_{2,2}=\begin{pmatrix} L_1 & P_2 \\ L_2 & P_2 \\ M & P_2 \end{pmatrix} \mathscr{M}_{2,3}=\begin{pmatrix} H & P_3 \end{pmatrix}$$

*The first columns of these matrices denote the configurations of $\varepsilon_F$ and the second* *denote the configurations of $\varepsilon_E$. The matrix $\mathscr{M}_{1,1}$ is applied when $\varepsilon_F$ requires to transform into $L_1$. If $L_1$ is activated in $\varepsilon_F$, then $P_1$ or $P_2$ of $\varepsilon_E$ can be activated for coherent behavior. The matrix $\mathscr{M}_{2,3}$ is applied when $\varepsilon_E$ applies $P_3$ to optimize the production. In this case, only the configuration $H$ of $\varepsilon_F$ is qualified to coordinate with $\varepsilon_E$.*

*Assume that the on-going distributed configuration of $Sys$ is $D'_\Gamma = (M, P_1)$. $\varepsilon_F$ and $\varepsilon_E$ send the following two signals simultaneously to the coordinator when the drill machine $Dr_1$ of $\varepsilon_F$ breaks down and $\varepsilon_E$ requires to improve its production:*

$$Request(From\_\varepsilon_F, To\_\varpi, Target\_L_2, Dr_1!),$$
$$Request(From\_\varepsilon_E, To\_\varpi, Target\_P_3, User).$$

*Accordingly, two* Judgement Matrices *$\mathscr{M}_{1,2}$ and $\mathscr{M}_{2,3}$ are applied. The row vector $(L_2, P_1)$ is finally selected by the coordinator,* i.e., *$D''_\Gamma = (L_2, P_1)$. Therefore, the request from $\varepsilon_E$ is rejected by the coordinator but the one from $\varepsilon_F$ is accepted.*

The number of subsystems that do not send reconfiguration requirements but need to apply local reconfigurations during a distributed reconfiguration process is denoted by $\beta_{rec}$. We model each distributed reconfiguration process by a TNCES. The module of the optimal coordination solution $D''_\Gamma$ is defined by $\beta_{req} + \beta_{rec}$ traces. A trace is a sequence of places and transitions, which corresponds to a coordination solution for a subsystem.

If a trace corresponds to the coordination solution of a subsystem $\varepsilon$ that does not send a reconfiguration requirement, then such a trace starts with a place $p$, ends with a place $p'$, and contains a transition $t$ between the two places, which is denoted by $ptp'$. The firing of $t$ means a reconfiguration command signal is sent to $\varepsilon$ by the coordinator. If a trace corresponds to the coordination solution of a subsystem $\varepsilon$ that sends a reconfiguration requirement, then such a trace starts with a place $p$, ends with a place $p'$, and contains three transitions between the two places, which is denoted by $pt_a/t_r/t_{rr}p'$. The firing of $t_a$ means that the reconfiguration requirement from $\varepsilon$ is accepted, the firing of $t_r$ means that the requirement is rejected, and the firing of $t_{rr}$ means that the requirement is rejected but $\varepsilon$ also should apply a local reconfiguration scenario.

**Example 12** *The TNCES-based model of the distributed reconfiguration process in Example 11 is shown in Fig. 5.5. The model of the selected row vector $D''_\Gamma = (L_2, P_1)$ has two traces: the first $p_6t_{10}/t_{11}/t_{12}p_7$ corresponds to the result of the reconfiguration requirement from $\varepsilon_F$ and the second $p_8t_{13}/t_{14}/t_{15}p_9$ corresponds to the result of the reconfiguration requirement from $\varepsilon_E$. The firing of $t_{10}$ (resp, $t_{13}$) means that the requirement from $\varepsilon_F$ (resp,*

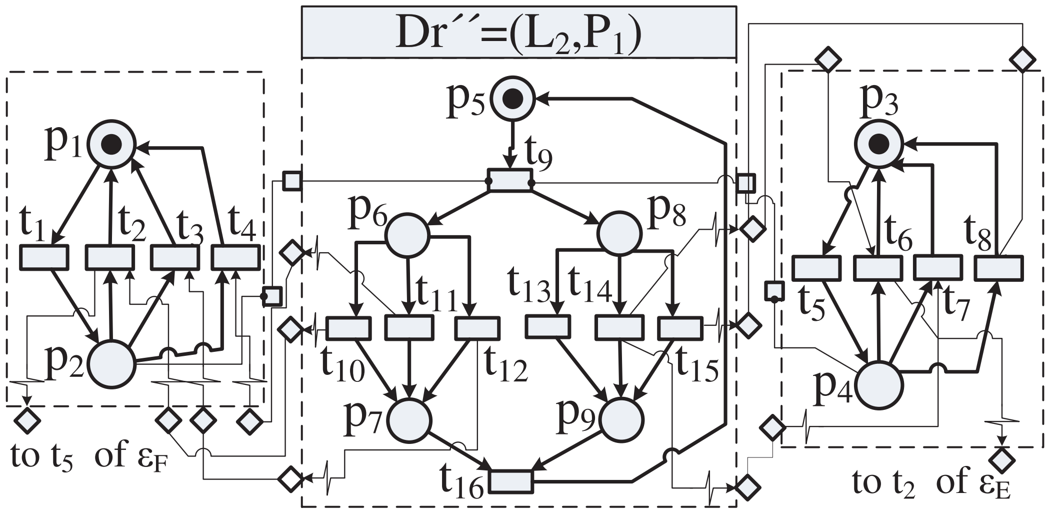


Fig 5.5 The TNCES model of a distributed reconfiguration process

*$\varepsilon_E$) is accepted, $t_{11}$ (resp, $t_{14}$) corresponds to the rejection of the requirement, and $t_{12}$ (resp, $t_{15}$) means that the requirement is rejected but the coordinator sends another command to it. Transition $t_8$ sends an event signal to $t_5$ of $\varepsilon_F$ in Fig. 5.2, which corresponds to the reconfiguration function changing $\varepsilon_F$ from $L_1$ to $M$. The required reconfiguration of $\varepsilon_E$ can be applied only after $t_6$ fires. However, in this distributed reconfiguration process, $\varepsilon_E$'s requirement is rejected. Thus, $t_2$ of $\varepsilon_E$ in Fig. 5.3 cannot fire and $\varepsilon_E$ cannot apply any reconfiguration scenario.*

## 5.4 SESA based Verification of DRDECSs

Model checking is a technique for automatically verifying the correctness properties of finite-state systems. Model checking for TNCESs and R-TNCESs is based on their reachability graph. The checked properties include the timely and correct response of the whole DRDECS to the concurrent reconfiguration requests and the valid behavior of subsystems after the application of a distributed reconfiguration scenario. All checked properties are specified by the temporal logic CTL or its extensions and are checked by the model-checker SESA.

For the R-TNCES models of subsystems, we focus on the following two cases: 1) if a reconfiguration command signal is received, whether a subsystem can respond and select a proper configuration, 2) after the implementation of a particular reconfiguration scenario, whether the new configuration can reflect correct functionalities and satisfy required temporal constraints.

**Example 13** The following CTL formula is proposed to the control module of $\varepsilon_F$ in Fig. 5.2:

$$z_0 \models AGAt_5XAFEt_9Xp_5$$

This formula is proven to be false by SESA. In fact, when $t_5$ fires, $\Gamma_{L_2}$ is selected. Thus, $t_9$ corresponding to $\Gamma_{L_1}$ cannot become enabled under this circumstance. The following formula has been proven to be true when $L_1$ is selected.

$$z_0 \models AGAt_4ORt_8XAFEt_9Xp_5$$

The eCTL formula below is applied to the TNCES $\Gamma_{L_1}$ of $\varepsilon_F$:

$$z_0 \models AGAt_{13}XAFEt_{15}ANDt_{32}XTRUE$$

This formula is proven to be false by SESA. Indeed, whenever $t_{13}$ fires, either $t_{15}$ or $t_{32}$ will eventually fire, which depends on the test result of the input workpiece. In particular, the following TCTL property is checked by SESA when $\varepsilon_F$ is in the behavior mode $M$:

$$z_0 \models EF[14, 24]p_{16} = 1$$

This formula has been proven to be true. In fact, in the medium production mode, the drill machine Dr1 or Dr2 can be activated in at least $14$ time units after the system starts.

For the TNCES model of a distributed reconfiguration process, SESA is applied to check the system behavior when a particular distributed reconfiguration scenario is applied by the coordinator. Indeed, we have to check after the coordination solution is obtained for the received concurrent reconfiguration requirements, whether all the relevant subsystems can react correctly.

**Example 14** We check the TNCES model in Example 12. The original distributed configuration is $D_\Gamma{}' = (L_1, P_1)$. We specify the following functional property according to the temporal logic CTL:

$$z_0 \models AGAt_{10}XEFEt_2XAFp_2$$

The formula is proven to be true. Firing $t_{10}$ means that the reconfiguration requirement of $\varepsilon_F$ is accepted by the coordinator. Firing $t_2$ means that $\varepsilon_F$ receives a command signal from the coordinator, which will trigger the firing of $t_5$ in $\varepsilon_F$, such that $\varepsilon_F$ is transformed from $L_1$ to $L_2$. The following formula is proven to be false.

$$z_0 \models AGAt_{13}XEFEt_7XAFp_9$$

In fact, the requirement from $\varepsilon_E$ is rejected by the coordinator according to the final coordination solution $D''_\Gamma = (L_2, P_1)$. Therefore, $\varepsilon_E$ should not apply any reconfiguration during this distributed reconfiguration process.

## 5.5 Discussion

The rate of exchanged messages is an important criterion in order to guarantee an acceptable level of safety and robustness in real-world industry such as wireless applications. In this section, the proposed solution for concurrent reconfiguration requirements is evaluated by counting the number of exchanged messages between subsystems and the coordinator.

Assume that a DRDECS has $\mathbb{n}$ subsystems and a coordinator. If no coordinator is applied to the DRDECS, any subsystem desiring a global reconfiguration has to inform all others before applying any global reconfiguration scenario. The concurrent reconfiguration requirements are treated one by one. However, if a coordinator is applied, a subsystem desiring a global reconfiguration only sends a requirement to the coordinator, where the reaction of the whole system is decided. All the concurrent requirements are treated according to the applied *Judgement Matrices*. It is assumed that in this discussion the rejected subsystems during a distributed reconfiguration process will send again the same requirements to the coordinator until they are accepted in the future and no new reconfiguration requirements arise before all the requirements are accepted.

We denote by $\beta_{mes}$ the number of messages when no coordinator is applied in a DRDECS. Let $\beta_{req}$ be the number of concurrent reconfiguration requirements that are accepted one by one and $\forall i \in \{1, 2, ..., \beta_{req}\}$, $\beta^i_{rec}$ be the number of final reconfigured subsystems for the $i^{th}$ requirement. If no coordinator is applied, then $\beta_{mes}$ is computed by:

$$\beta_{mes} = \sum_{i=1}^{\beta_{req}} 3\beta^i_{rec}(\mathbb{n} - 1)$$

In this case, the $\beta_{req}$ requirements are treated one by one. A subsystem desiring reconfigurations sends signals to all others before waiting their answers and deciding the reconfiguration scenario to be applied. Afterwards, feedback signals are sent to all others to broadcast its on-going configuration.

We use $\mathbb{s}$ to denote the steps to accept all the $\beta_{req}$ concurrent reconfiguration requirements when the proposed architecture is applied. We use $\beta^i_{tre}$ to denote the number of accepted requirements in the $i^{th}$ step ($i \in \{1, 2, ..., \mathbb{s}\}$). Then the number of exchanged messages, denoted by $\beta^c_{mes}$, is computed by:

$$\beta^c_{mes} = \sum_{i=1}^{\mathbb{s}} 2(\beta_{req} - \sum_{j=0}^{i-1} \beta^j_{tre} + \beta^i_{rec})$$

Indeed, $\beta_{req}$ subsystems desiring reconfigurations of corresponding devices send $\beta_{req}$ messages to the coordinator, but only the highest-priority message is accepted before a notification is sent to the rest (*i.e.*, *n-1*) subsystems. The coordinator decides any scenario to be applied once answers are received from the distributed subsystems.

To compare the two approaches, it is assumed that $\mathbb{n}$ subsystems send reconfiguration requirements simultaneously, *i.e.*, $\beta_{req} = \mathbb{n}$. Both approaches have the best case and the worst case.

#### 5.5.0.1 Communication without a coordinator

- Best Case: In this case, a subsystem desiring a reconfiguration sends a requirement to all the other subsystems but only itself needs to be reconfigured with the allowance of the other subsystems. Therefore, the number of exchanged messages in its best case is:

$$\beta^b_{mes} = 3\mathbb{n}(\mathbb{n} - 1)$$

- Worst Case: In this case, a subsystem sends a requirement to all the other subsystems and all the subsystems in the environment should apply local reconfigurations. Therefore, the number of exchanged messages in its worst case is:

$$\beta^w_{mes} = 3\mathbb{n}^2(\mathbb{n} - 1)$$

#### 5.5.0.2 Communication through a coordinator

- Best Case: In this case, all the $\mathbb{n}$ concurrent reconfiguration requirements are accepted, *i.e.*, $\mathbb{s} = 1$. There is a proper distributed reconfiguration function that satisfies all the requirements. Therefore, the number of exchanged messages in its best case is:

$$\beta_{mes}^{cb} = 4\mathbb{m}$$

- Worst Case: In this case, the concurrent reconfiguration requirements are accepted in $\mathbb{m}$ steps, *i.e.*, in each step, only one requirement is accepted but all the other subsystems need to be reconfigured. Therefore, the number of exchanged messages in its worst case is:

$$\beta_{mes}^{cw} = 3\mathbb{m}(\mathbb{m}+1)$$

Table 5.2 Comparison of exchanged messages

| $\mathbb{m}$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| $\beta_{mes}^{b}$ | 6 | 18 | 36 | 60 | 90 | 126 | 168 | 216 |
| $\beta_{mes}^{cb}$ | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |
| $\beta_{mes}^{w}$ | 12 | 54 | 144 | 300 | 540 | 882 | 1344 | 1944 |
| $\beta_{mes}^{cw}$ | 18 | 36 | 60 | 90 | 126 | 168 | 216 | 270 |

We compare the two results in TABLE 5.2. With the increase of the number of subsystems, the advantage of the developed approach is clearly shown.

## 5.6 Summary

This chapter presents a novel virtual coordinator for a DRDECS to deal with the reconfigurable coordination of a group set of reconfigurable discrete event control subsystems. These subsystems are modeled by R-TNCESs. Concurrent reconfiguration requirements are well solved by *Judgement Matrices* with an optimal coordination solution while the exchanged messages between subsystems and the coordinator are well-controlled.

# Chapter 6 Extended R-TNCESs

Traditionally, manufacturing is an energy-intensive process, using motors, steam, and compressed air systems to transform raw materials into durable goods and consumer products [91], [92], [93]. Recent research shows that switching machines of a manufacturing system into their energy-efficient modes when they are idle during production can make considerable contribution to the reduction of energy demand and thus can reduce carbon footprint as well as operating costs [94], [95], [96], [97], [98], [99], [100], [101], [102]. This chapter deals with the formal modeling and verification of reconfigurable and energy-efficient manufacturing systems (REMSs) that are considered as reconfigurable discrete event control systems (RDECSs).

## 6.1 Motivation

Taking the advantage of dynamic reconfigurations of machines of a reconfigurable manufacturing system (RMS) between their working modes and energy-efficient modes can reduce system energy consumption. In this chapter, an RMS with such energy-efficient operations is called a reconfigurable and energy-efficient manufacturing system (REMS). Such kind of systems can be abstracted as reconfigurable discrete event control systems (RDECSs) when only their logic behavior properties are investigated. In this chapter, a reconfiguration is called a local reconfiguration, if it is applied for switching a machine of an REMS between its working mode and energy-efficient mode. A reconfiguration is named a global reconfiguration if it is applied for switching an REMS between different configurations .

An REMS not only allows global reconfigurations for switching the system from one configuration to another, but also allows local reconfigurations on components for saving energy when the system is in a particular configuration. In addition, the un-reconfigured components of such a system should continue running during any reconfiguration. As a result, during a system reconfiguration, the system may have several possible pathes and may fail to meet control requirements if concurrent reconfiguration events and normal events are not controlled. It means that the uncontrolled concurrence of reconfiguration events and normal events may cause faults such as deadlocks and overflow [103], [104], [105], [106].

The formalism reconfigurable timed net condition/event system (R-TNCES) is a reconfigurable extension of the timed net condition/event system (TNCES). TNCESs have a visual graph expression, a clear modular structure, and an exact mathematical definition inherited from Petri nets. In addition, TNCESs have a strong analysis software tool: SESA. System behavior properties, such as state/event trajectories, and temporal requirements, can be specified by computation tree logic (CTL), extended CTL (eCTL), and timed CTL (TCTL) before checked by SESA automatically. If a property is satisfied by the system, the model checker will return a 'True'. Otherwise, a counterexample will be returned. Therefore, TNCESs has been widely applied in verification and validation of industry control systems especially for manufacturing systems. The verification of an R-TNCES can be performed with the assistance of SESA.

However, R-TNCESs cannot fully meet our requirements for an REMS. In an R-TNCES, reconfiguration functions model system reconfiguration events and transitions model normal events. However, the concurrence of reconfiguration functions and transitions is forbidden in an R-TNCES, which is in fact inconsistent with system requirements of REMSs. As a result, formal verification of such complex systems cannot be performed.

Motivated by the fact aforementioned, this chapter extends R-TNCESs. First, the reconfiguration functions of an R-TNCES are assigned with action ranges and concurrent decision functions. After that, they are divided into two types according to their action ranges: major and minor reconfiguration functions. The major ones are used to model global reconfiguration events, whereas the minor ones are applied to model local reconfiguration events. Accordingly, the dynamics of an R-TNCES is updated for these extensions such that the concurrence of reconfiguration events and normal events can be conditionally allowed to guarantee the system correctness. Afterwards, an implementation method for an extended R-TNCES is developed. Finally, the software tool SESA is applied to check system functional, temporal, and energy-efficient properties. An automatic assembly system is used to illustrate this work.

## 6.2 Reconfigurable and Energy-efficient Manufacturing Systems

This chapter treats a reconfigurable and energy-efficient manufacturing system (REMS) as a reconfigurable discrete event control system. This section presents system specification and interesting system dynamics before illustrating them with an automatic assembly system.

### 6.2.1 System Specification

An REMS is designed with a set of configurations to meet various requirements in different execution environments. A configuration $Con$ is defined as

$$Con = (Com, Str, Dat),$$

where $Com$ is a set of all activated components in $Con$, $Str$ defines the structure, *i.e.*, the the connection relationship and the communication protocol among components of $Com$, and $Dat$ denotes the set of all global variables and parameters of $Con$.

An REMS is denoted by

$$Sys = (\sum, R_c),$$

where $\sum$ is the set of $n$ configurations and $R_c : \sum \to \sum$ is the reconfigurable controller dealing with system reconfigurations.

There are two types of system reconfigurations in an REMS: global and local reconfigurations. The former are applied for switching system configurations. The latter are applied for switching an activated component between its working mode and energy-efficient mode when the system is in a particular configuration.

An REMS starts running as described in one of these configurations. After that, it should be able to change into other configurations smoothly due to the detection of component faults or other well-defined conditions. In addition, in each configuration, local reconfigurations can be applied to components such that the components can reconfigure themselves into their energy-efficient modes to save energy when they are idle and turn back to their working modes when the system needs them.

Dynamics of an REMS can be described by the evolution of system states. The evolution is caused by occurred events. An REMS includes normal events, local reconfiguration events, and global reconfiguration events.

1. If a normal event occurs, the system changes its state within its current configuration.

2. If a local reconfiguration event occurs, a component of current configuration switches into its energy-efficient mode or switches back into its working mode.

3. If a global reconfiguration event occurs, the system switches into another configuration.

Meanwhile, during a global or local reconfiguration, if normal events meet their occurring conditions and they are not modified by the occurred reconfiguration events, they should go on occurring. However, this kind of concurrence brings safety threat to the system, since they may cause unboundedness, deadlocks, even other functional or temporal failings.

### 6.2.2 Running Example

An automatic assembly system, denoted by AAS, is applied to illustrate works presented in this chapter. AAS includes three workstations ($W1$, $W2$, and $W3$) and four robots ($Rb1$, $Rb2$, $Rb3$, and $Rb4$). It is assumed that robots are high energy consumption machines. The respective time consumption of $W1$, $W2$, and $W3$ to finish a machining task is 40 time units, 30 time unites, and 50 time unites. The time consumption of both $Rb1$ and $Rb2$ to finish a task is 20 time units. The time consumption of both $Rb3$ and $Rb4$ to finish a task is 25 time units. The default working process diagram of AAS is shown in Fig. 6.1.

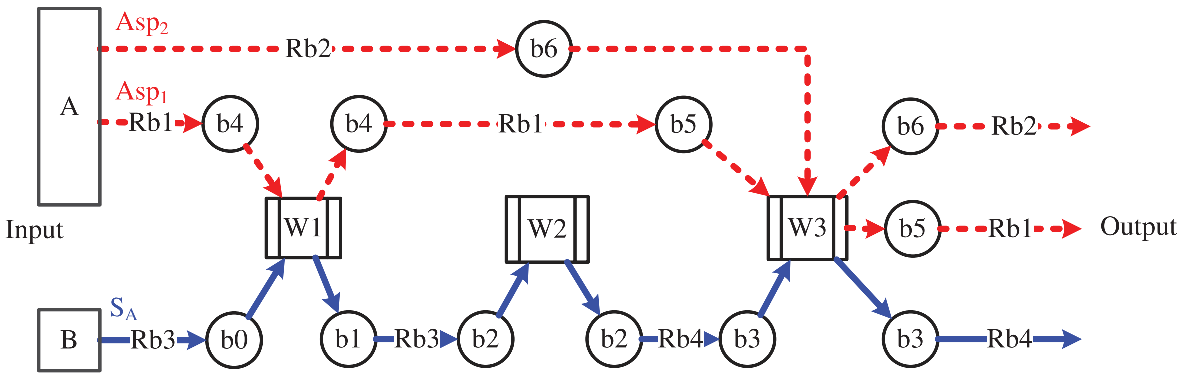


Fig 6.1 Default working process diagram of AAS

The main function of AAS is to assemble machine parts into a subassembly of a vehicle, to be marked by $S_A$. Robots $Rb1$ and $Rb2$ move machine parts from Input into AAS, transfer machine parts between workstations, and remove trashy machine parts to the Output. Dotted arrows in Fig. 6.1 are used to denote the movements of machine parts during an assembly process. On the other hand, $S_A$ is shifted along $W1$, $W2$, and $W3$ by robots $Rb3$ and $Rb4$. Solid arrows in Fig. 6.1 are used to denote the movement of $S_A$. To make it clear, $b0$, $b1$, ... and $b6$ are used to denote positions where machine parts or $S_A$ should be during an assembly process. The main assembly process is briefly described as follow three steps:

1. The to-be-worked subassembly $S_A$ is shifted from Input $B$ to $b0$ by $Rb3$. A machine part $Asp_1$ is delivered to $b4$ from the input $A$. After that, $Asp_1$ and $S_A$ are preprocessed on $W1$. Then, the preprocessed $S_A$ is moved to $b1$ automatically. The preprocessed $Asp_1$ is moved to $b4$ automatically before being moved to position $b5$ by $Rb1$.

2. $S_A$ is transported to $b2$ from $b1$ by $Rb3$. Then, a second preprocess for $S_A$ is done by $W2$. After that, $S_A$ is shifted to $b3$ from $b2$ by $Rb4$.

3. A machine part $Asp_2$ is delivered to $b6$ by $Rb2$. Then, $W3$ starts the assembly after $S_A$ is in $b3$, preprocessed $Asp_1$ is in $b5$, and $Asp_2$ is in $b6$. After the assembly, the machined

$S_A$ is moved out by $Rb4$. Two other trashy machine parts are removed out of AAS by $Rb1$ and $Rb2$, respectively.

It is assumed that four behavior modes are designed for AAS. Their work processes are illustrated as follows:

- $Mode1$: $Mode1$ is the default mode as depicted in Fig. 6.1, where all the robots are used.
- $Mode2$: $Mode2$ is a responding mode when $Rb2$ breaks down, where $Rb1$ should update itself to perform the function of $Rb2$.
- $Mode3$: $Mode3$ is applied when $Rb4$ breaks down during the execution of $Mode1$, where the work of $Rb4$ has to be done by $Rb3$.
- $Mode4$: $Mode4$ is applied when both $Rb1$ and $Rb2$ break down, where only $Rb1$ and $Rb3$ are applied. In this case, $Rb1$ should cover the function of $Rb2$ as in $Mode2$ and $Rb3$ should cover the function of $Rb4$ as in $Mode3$.

In each behavior mode, the applied robots should be able to reconfigure themselves into their energy-efficient modes when they are idle and reconfigure themselves back into their working modes when they have new tasks. A local reconfiguration for switching a robot from its working mode to its energy-efficient mode consumes one time unit. Likewise, a local reconfiguration for switching a robot from its energy-efficient mode back to its working mode consumes one time unit, as well.

To avoid the halt of a continuous production line, possible dynamic reconfigurations applied for switching AAS between these behavior modes are shown in Fig. 6.2. The solid arrows denote global reconfigurations and dotted ones denote local reconfigurations.

It is assumed that a robot consumes one energy unit per time unit when it works in its working mode. However, it only consumes $30\%$ energy units per time unit when it works in its energy-efficient mode. Note that the numerical value '$30\%$' is an assumption by the authors to facilitate the quantitative analysis on energy-efficient operations. It does not come from any literature on industry systems.

Obviously, the possible reconfiguration events of AAS can occur simultaneously with many normal events in it. For example, when $Rb1$ is being modified by a global reconfiguration or being switched into its energy-efficient mode, only its own work needs to stop for a while and the workstations and other running robots should do their jobs unaffectedly.

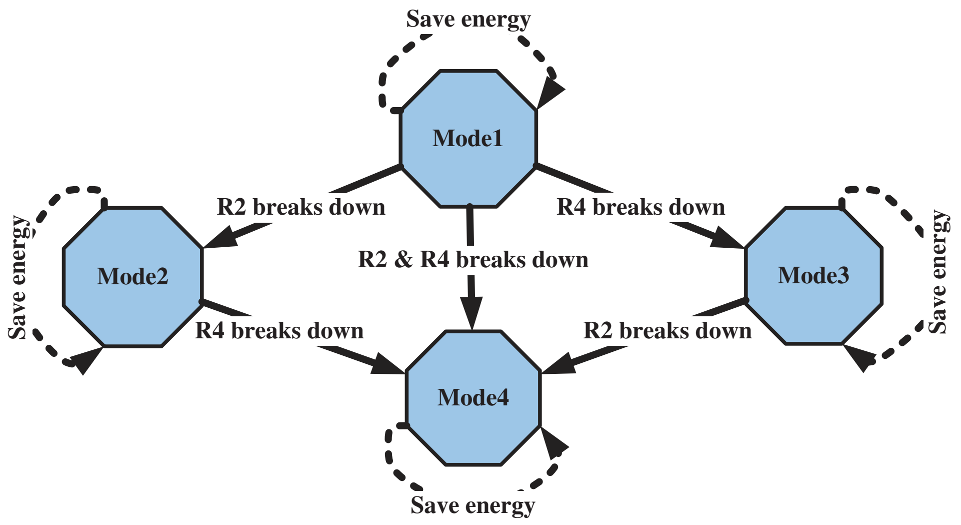


Fig 6.2 Possible reconfigurations in AAS

## 6.3 Extended R-TNCESs

The formalism reconfigurable timed net condition/event system (R-TNCES)is an extension of the timed net condition/event systems (TNCES). Reconfiguration functions of R-TNCESs can be used to model global reconfiguration events of REMSs. However, they are not proper to model local reconfiguration events of REMSs directly. In addition, the concurrence of normal events and reconfiguration events is currently not allowed in an R-TNCES. Therefore, in order to perform correct formal verification of a REMS, this chapter extends R-TNCESs. This section analyzes the drawbacks of R-TNCESs on investigating REMSs before represents the proposed extended R-TNCESs.

### 6.3.1 Drawbacks of R-TNCESs

The TNCES models for the four behavior modes of AAS are denoted by $\Gamma_1 = (N_{\Gamma_1}, z1_0)$, $\Gamma_2 = (N_{\Gamma_2}, z2_0)$, $\Gamma_3 = (N_{\Gamma_3}, z3_0)$, and $\Gamma_4 = (N_{\Gamma_4}, z4_0)$, respectively. The set of all possible reconfiguration events of AAS is marked by $\mathcal{R}$=$\{r_{1,s}$, $r_{2,s}$, $r_{3,s}$, $r_{4,s}$, $r_{1,s}^{-1}$, $r_{2,s}^{-1}$, $r_{3,s}^{-1}$, $r_{4,s}^{-1}$, $r_{1,2}$, $r_{1,3}$, $r_{1,4}$, $r_{2,4}$, $r_{3,4}\}$. The reconfiguration event $r_{i,s}$ indicates a local reconfiguration that transforms robot $Rbi$ into its energy-efficient mode and $r_{i,s}^{-1}$ is the reverse of $r_{i,s}$, *i.e.*, to transform robot $Rbi$ from its energy-efficient mode into its working mode. The implementation of the events $r_{i,s}$ and $r_{i,s}^{-1}$ does not change the current behavior mode but can switch robot $Rbi$ between its working mode and energy-efficient mode according to its busy/idle status and waiting time. Finally, $r_{i,j}$ $(i \neq j)$ denotes a global reconfiguration event that transforms AAS from the configuration Mode$i$ into Mode$j$.

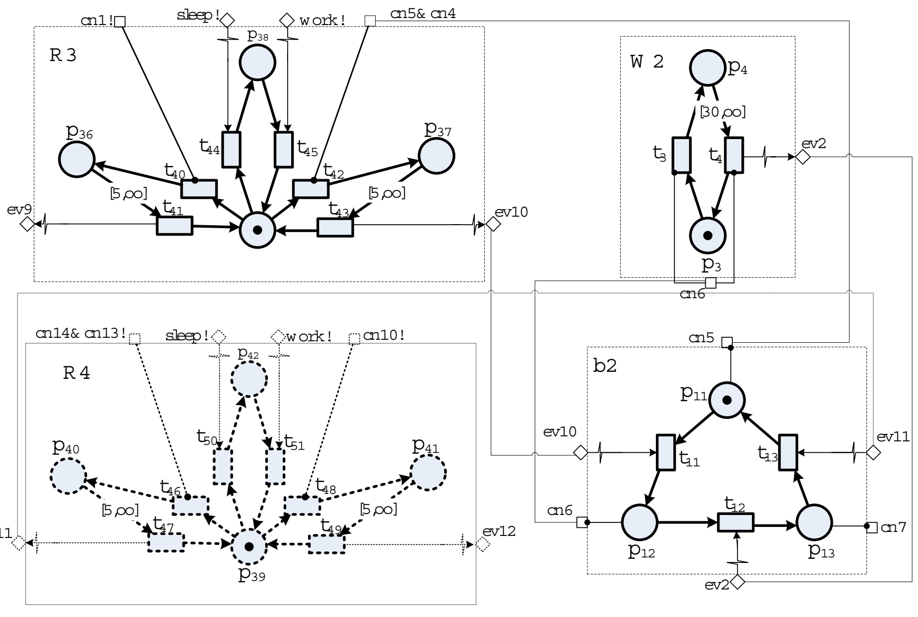


Fig 6.3 TNCES-based model of $R3$, $R4$, and $W2$ in Mode1

The firing of a reconfiguration function of an R-TNCES changes the system configuration. As a consequence, if reconfiguration functions are applied to model local reconfiguration events for switching components between their working modes and energy-efficient modes directly, the number of system configurations should be enlarged. For example, the configuration Mode4 should be considered as four different configurations: 1) Both $Rb1$ and $Rb3$ are in their working modes, 2) $Rb1$ is in working mode and $Rb3$ is in energy-efficient mode, 3) $Rb3$ is in working mode and $Rb1$ is in energy-efficient mode, and 4) Both $Rb1$ and $Rb3$ are in their energy-efficient modes. These four configurations are with the same structure. However, they should be verified separately. Obviously, this increases the verification cost and burdens the whole design process.

Generally, transitions in an R-TNCES model normal events of a reconfigurable discrete event control system. Whereas, reconfiguration functions are used to model system reconfiguration behavior. However, the concurrence of reconfiguration functions and transitions is not allowed in R-TNCESs, which is in fact inconsistent with requirements of REMSs. To make it clearer, let us take the modules $Rb3$, $Rb4$, and $W2$ as an example. Their TNCES-based models in Mode1 and Mode4 are shown in Figs. 6.3 and 6.4, respectively. The differences between them are marked by dotted lines.

**Example 15** *Suppose that a reconfiguration function $r_{1,4}$ gets enabled at state $S3$ when AAS is in Mode1. The physical meaning of $S3$ is that 1) $Rb3$ just finishes transporting $S_A$ to $b2$ and 2) $W2$ is ready to process $S_A$. Assume that at this time a fault is detected in $Rb4$.*

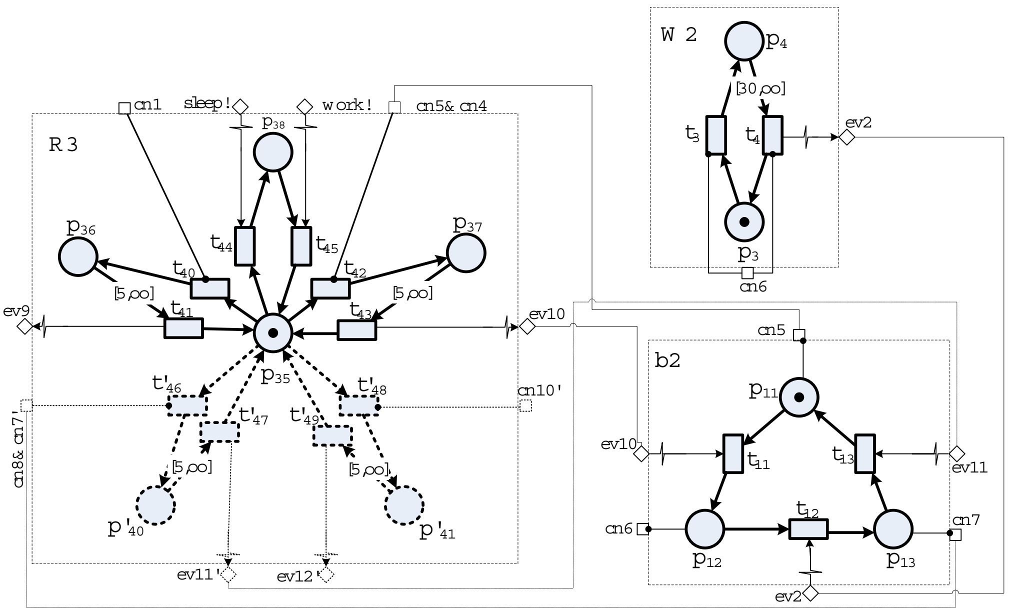


Fig 6.4 TNCES-based model of $R3$, $R4$, and $W2$ in Mode4

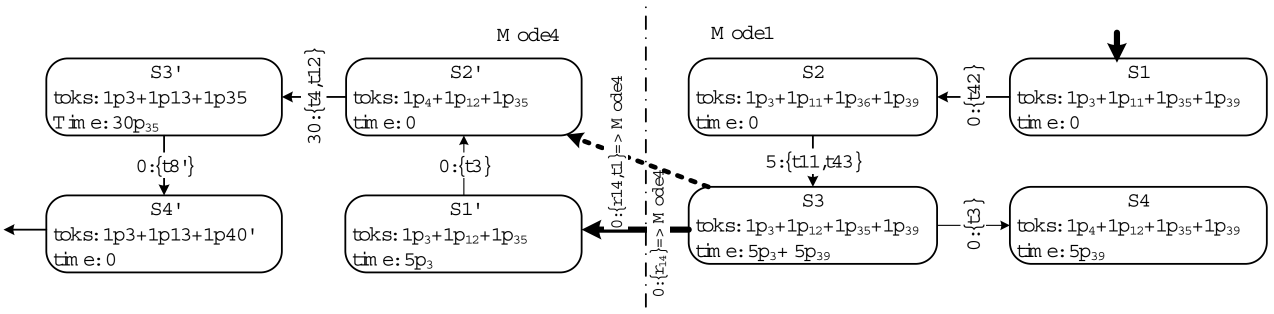


Fig 6.5 State transition graph of Example 1

*$Rb4$ should be removed. Meanwhile, $Rb3$ must update itself soon in order to cover $Rb4$'s task. According to the design requirements for AAS, $W2$ should go on working 'naturally' at this time,* i.e.*, the enabled transition $t_3$ can fire at this state. However, the concurrence of reconfiguration functions and transitions is not allowed in R-TNCESs. Therefore, at state $S3$, only $r_{1,4}$ fires alone and AAS turns to the state $S1'$. Afterwards, $t_3$ fires, which leads to the next state $S2'$. However, if $r_{1,4}$ and $t_3$ fire together, AAS turns to the state $S2'$ directly without generating $S1'$. The state transition graph of this case is shown in Fig. 6.5.*

**Example 16** *Assume that two reconfiguration functions $r_{3,s}$ and $r_{4,s}$ get enabled simultaneously at state $S4$. The physical meaning of $S4$ is that 1) $W2$ just starts its work and 2) both $Rb3$ and $Rb4$ are idle. The firing of $r_{3,s}$ and $r_{4,s}$ only changes the states inside their modules, but neither alter the system structure nor enable/disable any other transitions outside. That*

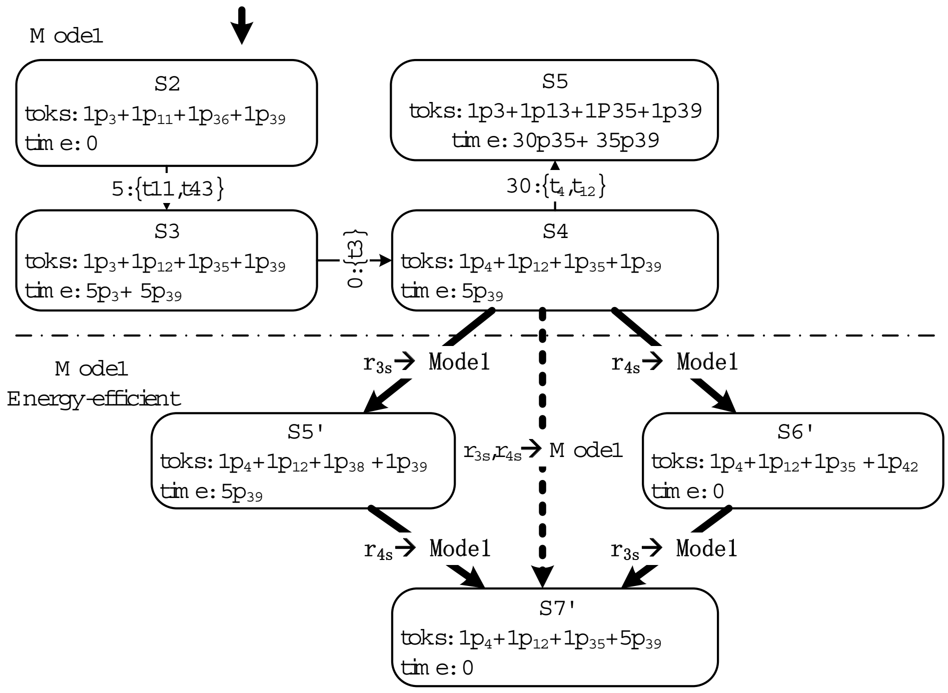


Fig 6.6 State transition graph of Example 2

*is to say, the firing of* $r_{3,s}$ *and* $r_{4,s}$ *does not change the current system configuration. According to the design requirements for AAS, both* $Rb3$ *and* $Rb4$ *can reconfigure themselves into energy-efficient modes freely when they will be idle for more than two time units. However, the concurrence of multiple reconfiguration functions is neither allowed in R-TNCESs. Therefore, at state* $S4$*, only* $r_{4,s}$ *or* $r_{3,s}$ *fires alone. After that, the rest one fires since it is still enabled. However, if* $r_{3,s}$ *and* $r_{4,s}$ *fire together, AAS turns to the state* $S7'$ *directly. The state transition graph of this case is shown in Fig. 6.6.*

In conclusion, the original R-TNCES formalism is not sufficient to model an REMS. The reason can be explained from the following three aspects:

- Reconfigurations at the component level only change component behavior modes between their working modes and energy-efficient modes rather than changing system configurations. If this kind of reconfigurations is modeled by reconfiguration functions directly, the number of system configurations should be enlarged, which increases the verification cost and burdens the whole design process.

- The concurrence of reconfiguration functions and transitions is not allowed in R-TNCESs. However, from the above examples, the concurrence of reconfiguration events and normal events is a common phenomenon in an REMS.

- Since the local reconfigurations for energy-efficient operations cannot be properly described, the corresponding dynamics of them and reasonable analysis cannot be performed.

To this end, this chapter extends R-TNCESs to achieve two aims. First, all possible events including concurrent events that may occur in REMSs can be properly described. Second, the concurrence of reconfiguration functions and transitions should be controlled to ensure the system correctness.

## 6.3.2 Extended R-TNCESs

An extended R-TNCES has the same structure with the original R-TNCES. It is composed of a behavior module and a control module, denoted by $eRN = \{\mathcal{B}, \mathcal{R}\}$. The definition of system states is not changed, as shown in Chapter 4. In the extended R-TNCES, reconfiguration functions are newly assigned with action ranges and concurrent decision functions. In addition, the firing rules of transitions and reconfiguration functions are updated such that they are conditionally allowed to fire concurrently.

### 6.3.2.1 Modified Reconfiguration Functions

In order to model the two types of reconfiguration events in an REMS directly, A concept namely *action range* is developed for each reconfiguration function of an R-TNCES. In addition, a *concurrent decision function* is also assigned to a reconfiguration function to constrain concurrent transitions that may lead to undesired states such as deadlocks and overflow during a reconfiguration. For the sake of brevity, a reconfiguration function indicates a modified reconfiguration function in what follows.

**Definition 5** *A reconfiguration function $r$ of an extended R-TNCES $eRN$ is a structure $r$=($Cond$, $s$, $x$, $\Lambda$, $\Pi$). $Cond \to \{true, false\}$ is the pre-condition of $r$. $s$: $\Omega \to \Omega$ is the structure modification instruction. $x$: $R(N_{\Gamma i}, z_{0i}) \to \mathscr{Z}_{0j}$ is the state correlation function, where $\mathscr{Z}_{0j}$ is a set of feasible initial states of $\Gamma_j$. ${}^{\star}r = \Gamma_i = (N_{\Gamma i}, z_{0i})$ (*resp*, $r^{\star} = \Gamma_j = (N_{\Gamma j}, z_{0j})$) denotes the TNCES before (*resp*, after) $r$ fires. $\Lambda \in (N_{\Gamma i} \cup N_{\Gamma j})$ denotes the action range of $r$. $\Pi(r, \mathcal{Z}) \to T$ is a concurrent decision function deciding a set of forbidden transitions that cannot fire together with $r$ at state $\mathcal{Z}$.*

The reconfiguration functions of extended R-TNCESs are divided into two types: major and minor reconfiguration functions. For a reconfiguration function $r$=($Cond$, $s$, $x$, $\Lambda$, $\Pi$) with ${}^{\star}r = \Gamma_i = (N_{\Gamma i}, z_{0i})$ and $r^{\star} = \Gamma_j = (N_{\Gamma j}, z_{0j})$, it is a major reconfiguration function if and only if $N_{\Gamma i} \neq N_{\Gamma j}$. Otherwise, it is a minor reconfiguration function. Let $\mathcal{R}_{ma}$ and $\mathcal{R}_{mi}$

denote the sets of major and minor reconfiguration functions of $eRN$, respectively. Then we have $\mathcal{R} = \mathcal{R}_{ma} \cup \mathcal{R}_{mi}$ and $\mathcal{R}_{ma} \cap \mathcal{R}_{mi} = \emptyset$.

The implementation (firing) of a major reconfiguration function changes the structure of the current activated TNCES, whereas the implementation (firing) of a minor reconfiguration function only adjusts partial states of the activated TNCES within its action range.

Similar to Petri nets, the 'conflict' concept is proposed for two enabled reconfiguration functions. We have the following two cases:

1. For two reconfiguration functions within the same type, *i.e.*, both are minor or major reconfiguration functions, if their action ranges have intersections, they are conflicting.

2. For a minor reconfiguration function and a major reconfiguration function, if the action range of the minor reconfiguration function is not completely covered by that of the major reconfiguration function, they are conflicting.

If two reconfiguration functions are conflicting, they cannot be implemented simultaneously. The symbol $r_1 \| r_2$ denotes that reconfiguration functions $r_1$ and $r_2$ are not conflicting.

Similar to the definition of *steps* in TNCESs, an *r*-step in an extended R-TNCES is a maximal set of reconfiguration functions that can fire simultaneously at a particular state. An *r*-step should satisfy the following two conditions:

1. For any two reconfiguration functions $r_i$ and $r_j$ ($r_i \neq r_j$) in an *r*-step $\gamma$, $r_i$ and $r_j$ are not conflicting, *i.e.*, $r_i \| r_j$.

2. There does not exist any other maximal set of reconfiguration functions $\gamma'$ such that $\gamma \subset \gamma'$.

Accordingly, two *r*-steps $\gamma_1$ and $\gamma_2$ are conflicting, if $\exists r_1 \in \gamma_1, \exists r_2 \in \gamma_2, r_1 \neq r_2$, $r_1$ and $r_2$ are conflicting.

#### 6.3.2.2 Dynamics of Extended R-TNCESs

Suppose that at state $\mathcal{Z} = \lceil N_\Gamma, z \rfloor$, multiple reconfiguration functions get enabled, to be denoted by

$$R^* = \gamma_1 \cup \gamma_2 \cup ... \cup \gamma_g,$$

where $\gamma_i$ ($i \in [1, g]$) is a maximal *r*-step at $\mathcal{Z}$ and $\forall i, j \in [1, g], i \neq j$, $\gamma_i$ and $\gamma_j$ are conflicting. At the same state $\mathcal{Z}$, the set of all enabled transitions is denoted by

$$T^* = u_1 \cup u_2 \cup ... \cup u_k,$$

where $u_i$ ($i \in [1, k]$) is a maximal step and $\forall i, j \in [1, k], i \neq j$, $u_i$ and $u_j$ are conflicting.

As a consequence, different compositions of *r*-steps and steps can occur simultaneously at this state. Given an enabled reconfiguration function $r$, we use $\mathscr{D}.T$ (resp, $\mathscr{D}.P$) to denote the set of deleted transitions (resp, deleted places) and $\mathscr{A}.T$ (resp, $\mathscr{A}.P$) to denote the set of added transitions (resp, added places) by firing it, where ${}^{\star}r = \Gamma_i = (N_{\Gamma i}, z_{0i})$, $r^{\star} = \Gamma_j = (N_{\Gamma j}, z_{0j})$, $N_{\Gamma i}$=($P_i$, $T_i$, $F_i$, $CN_i$, $EN_i$, $em_i$, $DC_i$), and $N_{\Gamma j}$=($P_j$, $T_j$, $F_j$, $CN_j$, $EN_j$, $em_j$, $DC_j$). We have the following two cases:

1) For a transition $t \in T_i$, if it is enabled simultaneously with a minor reconfiguration function $r = (Cond, s, x, \Lambda, \Pi)$ at state $\mathscr{Z}$=$\lceil N_{\Gamma_i}, z \rfloor$ and $t \notin \Lambda$, then $t$ can fire simultaneously with $r$, *i.e.*, $t \notin \Pi(r, \mathscr{Z})$.

2) For a transition $t \in T_i$, if it is enabled simultaneously with a major reconfiguration function $r = (Cond, s, x, \Lambda, \Pi)$ at state $\mathscr{Z}$=$\lceil N_{\Gamma_i}, z \rfloor$, then we have the following two subcases:

1. A spontaneous transition $t$ is forbidden to be concurrent with $r$ at $\mathscr{Z}$, if it meets one of the following conditions

   - If it is deleted by $r$, *i.e.*, $t \in \mathscr{D}.T$, it is forbidden by $r$, *i.e.*, $t \in \Pi(r, \mathscr{Z})$.
   - If $t \notin \mathscr{D}.T$ and all its elements are not changed by firing $r$, then it is allowed to fire simultaneously with $r$. Formally, if ${}^{\bullet}t_i ={}^{\bullet} t_j$, $t_i^{\bullet} = t_j^{\bullet}$, ${}^{-}t_i ={}^{-} t_j$, ${}^{\sim}t_i ={}^{\sim} t_j = \emptyset$, and $em(t)_i = em(t)_j$, we have $t \notin \Pi(r, \mathscr{Z})$.
   - If $t \notin \mathscr{D}.T$, some of its elements are modified by $r$, which include its preset, postset, source places, and firing mode, and we have the following two cases:
     (a) The preset, source places and firing mode of $t$ decide whether $t$ is enabled after the firing of $r$. Therefore, if its preset, source places, or firing mode will be changed by $r$, it can fire simultaneously with $r$. Formally, if ${}^{\bullet}t_i \neq {}^{\bullet}t_j$, ${}^{-}t_i \neq {}^{-}t_j$ or $em(t)_i \neq em(t)_j$, then $t \notin \Pi(r, \mathscr{Z})$.
     (b) The postset of $t$ does not change its enabling condition, but influences the structure of the net. Therefore, it is forbidden by $r$. Formally, if $t^{\bullet}{}_i \neq t^{\bullet}{}_j$, we have $t \in \Pi(r, \mathscr{Z})$.

2. A forced transition $t$ is forbidden to be concurrent with $r$ at $\mathscr{Z}$, if it further meets one of the following conditions:

   - Its firing mode is $\boxed{\nabla}$ and all of its forcing transitions are forbidden to be concurrent with $r$, *i.e.*, if $em_i(t)$=$em_j(t)$=$\boxed{\nabla}$ and $\forall t' \in {}^{\sim}t$, $t' \notin \Pi(r, \mathscr{Z})$, then $t \in \Pi(r, \mathscr{Z})$.

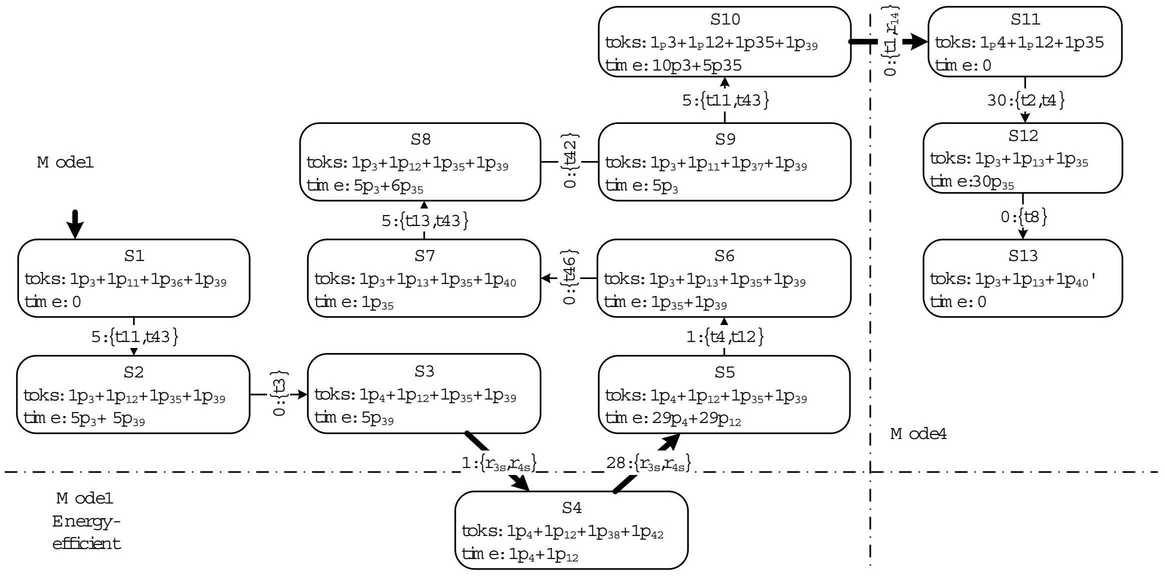


Fig 6.7 A fragment of the reachability graph of Example 3

- Its firing mode is ⍍ and at least one of its forcing transitions is forbidden by $r$, *i.e.*, if $em_i(t)$=$em_j(t)$=⍍ and $\exists t' \in {}^{\sim}t$, $t' \notin \Pi(r, \mathcal{Z})$, then $t \in \Pi(r, \mathcal{Z})$.

Since an extended R-TNCES allows the concurrence of multiple reconfiguration functions and transitions, the reachability graph of an extended R-TNCES is defined as follows:

**Definition 6** *The reachability graph of an extended R-TNCES $eRN$ is a combination of several labeled directed graphs whose nodes are the states of $eRN$ and whose arcs are of three kinds: steps,* r*-steps, and combinations of a step and an* r*-step.*

- *The arc from state $\lceil N_{\Gamma i}, z_i \rfloor$ to state $\lceil N_{\Gamma i}, z'_i \rfloor$ is denoted by a step $u$ represented by: $\lceil N_{\Gamma i}, z_i \rfloor [u\rangle \lceil N_{\Gamma i}, z'_i \rfloor$, where $z'_i \in R(N_{\Gamma i}, z_i)$.*

- *The arc from state $\lceil N_{\Gamma i}, z_i \rfloor$ to state $\lceil N_{\Gamma j}, z_j \rfloor$ is labeled with an* r*-step $\gamma$ represented by: $\lceil N_{\Gamma i}, z_i \rfloor [\gamma\rangle \lceil N_{\Gamma j}, z_j \rfloor$. If $\gamma$ contains major reconfiguration functions, $N_{\Gamma i} \neq N_{\Gamma j}$. Otherwise, we have $N_{\Gamma i} = N_{\Gamma j}$ and $z_j \notin R(N_{\Gamma i}, z_i)$.*

- *The arc from $\lceil N_{\Gamma i}, z_i \rfloor$ to state $\lceil N_{\Gamma j}, z_j \rfloor$ is labeled with a step and an* r*-step $\{\mathcal{R}, u\}$ represented by: $\lceil N_{\Gamma i}, z_i \rfloor [\gamma, u\rangle \lceil N_{\Gamma j}, z_j \rfloor$. If $\gamma$ contains major reconfiguration functions, $N_{\Gamma i} \neq N_{\Gamma j}$. Otherwise, we have $N_{\Gamma i} = N_{\Gamma j}$.*

Obviously, the graphical representation of an extended R-TNCES model is the same with that of an R-TNCES model. However, system dynamics get enriched along with the

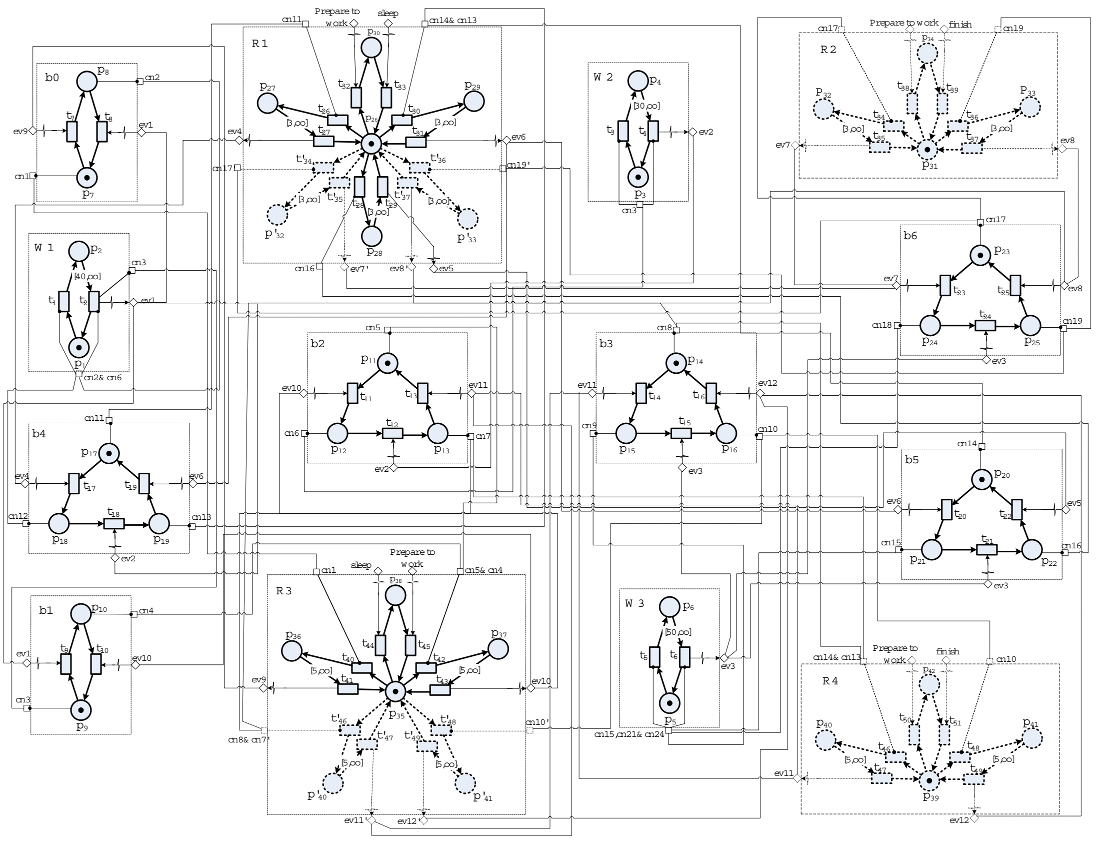


Fig 6.8 Behavior module of $eRN_{AAS}$

changes of reconfiguration functions. If we use an extended R-TNCES to model AAS, the graphical TNCES models shown in Figs. 6.3 and 6.4 are still correct. However, their reachability graphes get enriched during a same reconfiguration.

**Example 17** *A fragment of the reachability graph of the extended R-TNCES-based model of the example composed of $Rb3$, $Rb4$, and $W2$ is shown in Fig. 6.7. AAS starts running in $Mode1$. When it arrives at the state $S3$, two minor reconfiguration functions $r_{3,s}$ and $r_{4,s}$ get enabled and fire simultaneously to reconfigure robots $Rb3$ and $Rb4$ into their energy-efficient modes. After 28 time units, they reconfigure back to working modes. Assume that $R4$ is detected to have a fault at state $S10$, the major reconfiguration function $r_{1,4}$ gets enabled. At the meantime, $t3$ gets enabled simultaneously with $r_{1,4}$. Therefore, $t3$ fires simultaneously with $r_{1,4}$, which leads the transformation of AAS into $Mode4$.*

## 6.4 Verification of Extended R-TNCESs

In order to perform correct formal verification of AAS, an extended R-TNCES-based model should be built for it. The extended R-TNCES based model of AAS is marked by

$eRN_{AAS}$=$\{\mathcal{B}, \mathcal{R}\}$, $\mathcal{B}$=$\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$, $\mathcal{R}$=$\mathcal{R}_{ma}$∪$\mathcal{R}_{mi}$, $\mathcal{R}_{ma}$=$\{r_{1,2}$, $r_{1,3}$, $r_{1,4}$, $r_{2,4}$, $r_{3,4}\}$, and $\mathcal{R}_{mi}$=$\{r_{1,s}$, $r_{2,s}$, $r_{3,s}$, $r_{4,s}$, $r^{-1}_{1,s}$, $r^{-1}_{2,s}$, $r^{-1}_{3,s}$, $r^{-1}_{4,s}\}$. We have $\Omega$=$\{N_{\Gamma_1}$, $N_{\Gamma_2}$, $N_{\Gamma_3}$, $N_{\Gamma_4}\}$. The four major reconfiguration functions are conflicting with each other. The minor reconfiguration functions $r_{i,s}$ and $r^{-1}_{i,s}$ ($i \in [1, 4]$) are conflicting but others are not. The behavior module of $eRN_{AAS}$ is shown in Fig. 6.8, where elements drawn by dotted lines are possibly modified during the implementation of a major reconfiguration function. In order to apply automatic model checking to an extended R-TNCES, a TNCES-based nested state machine is developed to implement its control module.

### 6.4.1 Implementation of Extended R-TNCESs

First of all, major reconfiguration functions are grouped according to their action ranges. A set of state machines specified by TNCESs, which are called $Major_changer$s, is defined. Each state machine corresponds to a group of major reconfiguration functions that share the same action range. In a particular $Major\_changer$, each transition corresponds to a major reconfiguration function. The transitions in a state machine cannot fire simultaneously, which means that these modeled major reconfiguration functions by one state machine are conflicting with each other. Firing a transition $st$ in a $Major\_changer$ implies that a major reconfiguration function is implemented. A $Major\_changer$ is formalized as follows:

$$Major\_changer=(P, T, F, V, z_0),$$

where $\forall t \in T$, $|{}^\bullet t|$=$|t^\bullet|$=1, $\sum M_0(P)$=1, which means that only one place in $P$ owns a token at the initial state, and $V : T \rightarrow \{\boxed{\vee}\}$. The pre-condition $Cond$ can be modeled by input event/condition signals from external to transitions in a $Major\_changer$.

In addition, an actuator denoted by $Actuator$ is defined for each place $sp$ in all $Major_changer$s, which is marked by $Actuator$=$Act(sp)$. Each actuator is composed of a place $ap$ and a transition $at$ only, where ${}^\bullet ap$=$ap^\bullet$=$\{at\}$, ${}^\bullet at$=$at^\bullet$=$\{ap\}$, and $M(ap) = 1$. When the place $sp$ in a $Major\_changer$ receives a token, the actuator $Actuator$=$Act(sp)$ is activated. An $Actuator$ is formalized as follows:

$$Actuator=(P, T, F, V, z_0),$$

where $|P| = |T| = 1$, ${}^\bullet at = at^\bullet = \{ap\}$, ${}^\bullet ap = ap^\bullet = \{at\}$, $m_0(P) = 1$, and $V : T \rightarrow \{\boxed{\vee}\}$.

Similar to major reconfiguration functions, minor reconfiguration functions are grouped according to their action ranges. A set of state machines specified by TNCESs, which are called $Minor_changer$s, is defined. Each state machine corresponds to a group of

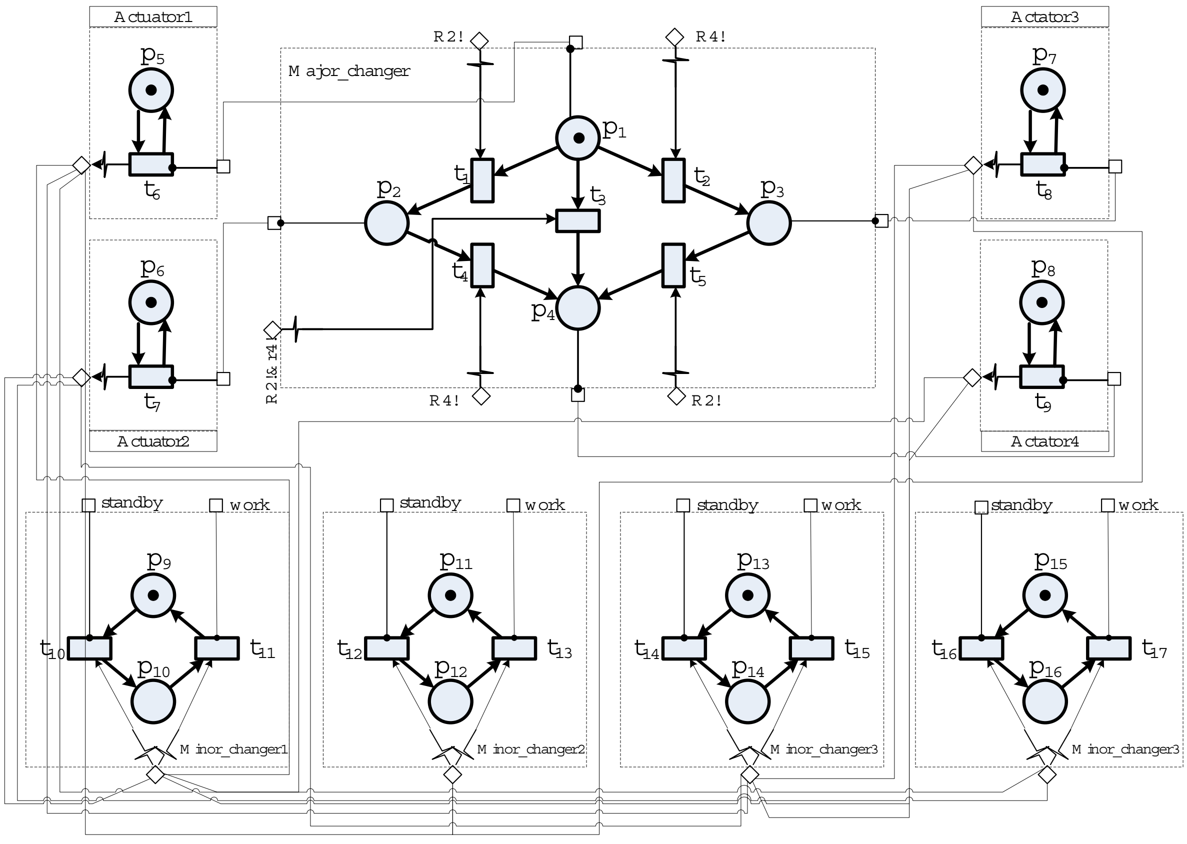


Fig 6.9 Control module of $eRN_{AAS}$

minor reconfiguration functions. If the action ranges of two minor reconfiguration functions are the same, they are modeled by transitions in a $Minor\_changer$. If the action range of a group of minor reconfiguration functions, to be modelled by a $Minor\_changer$, is completely covered by that of a group of major reconfiguration functions, to be modeled by a $Major\_changer$, then this $Minor\_changer$ can be activated only when this $Major\_changer$ is activated.

A $Minor\_changer$ is formalized as follows:

$$Minor\_changer\text{=}(P, T, F, V, z_0),$$

where $\forall t \in T$, $|{}^{\bullet}t|\text{=}|t^{\bullet}|\text{=}1$, $\sum M_0(P)\text{=}1$, which means that only one place in $P$ owns a token at the initial state, and $V : T \rightarrow \{\boxed{\vee}\}$. The pre-condition $Cond$ can be modeled by input event/condition signals from external to transitions in a $Minor\_changer$.

**Example 18** *Fig. 6.9 depicts the TNCES-based control module of $eRN_{AAS}$. It has only one $Major\_changer$, since the four major reconfiguration functions share the same action range. It has four $Minor\_changers$, since the four robots have four distinguished action ranges. The places $p_1$, $p_2$, $p_3$, and $p_4$ in $Major\_changer$ correspond to Mode1, Mode2,* *Mode3, and Mode4, respectively. When $t_3$ fires, the major reconfiguration function $r_{1,4}$ is implemented. Robots $Rb3$ and $Rb1$ are applied in every mode of AAS. Therefore, minor reconfiguration functions that transform them between energy-efficient modes and working modes are activated in every system behavior mode. Moreover, it is possible for them to fire simultaneously with other major reconfiguration functions.*

### 6.4.2 Formal Verification of AAS

Since the time when a major reconfiguration function can get enabled and fire cannot be predicted, this chapter applies an instruction insertion method to simulate AAS. In addition, $eRN_{AAS}$ evolves according to fired maximal steps and *r*-steps. Assume that AAS should finish 100 subassemblies. It starts with Mode1. At time $t_1$ when it finishes the $60^{th}$ subassembly, it reconfigures into Mode2 due to the fault detection of $Rb2$. Then, it goes on working in Mode2. At time $t_2$ when the $91^{st}$ subassembly is being processed, it transforms into Mode4 according to the fault detection of $Rb4$. During the whole process, minor reconfigurations, *i.e.*, transforming robots between their working modes and energy-efficient modes, are applied.

SESA is applied to compute the reachability graph of this whole process. A minimal path regarding time consumption from the initial state to the objective state is computed in each mode. In Mode1, it generates 23044 states, which takes 6990 time units to finish assembly of the first 60 subassemblies in the minimal path. In Mode2, it generates 85259 states, which costs 4127 time units to finish assembling the next 30 subassemblies in the minimal path. Finally, in Mode4, it generates 195007 states, which takes 1525 time units to finish assembling the last 10 subassemblies in the minimal path. Note that, two states can be considered to be same if and only if they have the same token numbers and time status.

Since each TNCES-based model of the behavior modes of AAS is a well-designed control system, they are proved to be qualified according to SESA, where eCTL based functional properties and TCTL based temporal properties are checked. In addition, the following eCTL formula is applied to the control module of $eRN_{AAS}$:

$$Z_0 = EX < t4ANDt12 > X < p12 = 1 >$$

This formula is proved to be false by SESA. The transition $t_{12}$ corresponds to the minor reconfiguration function $r_{2,s}$. Therefore, it can fire only when AAS is in Mode1 or Mode2. The following formula is proved to be true.

$$Z_0 = EX < t2ANDt10 > X < p10 = 1 >$$

Table 6.1 Time of robots on their energy-efficient modes

| Configuration | Mode1 | | | | Mode2 | | | Mode4 | |
|---|---|---|---|---|---|---|---|---|---|
| System Uptime | 6690 | | | | 4127 | | | 1525 | |
| Robot | Rb1 | Rb2 | Rb3 | Rb4 | Rb1 | Rb3 | Rb4 | Rb1 | Rb3 |
| Time on energy-efficient mode | 3233 | 4455 | 3818 | 2643 | 1004 | 2458 | 2428 | 523 | 435 |

Table 6.2 Energy consumption of robots

| Configuration | Mode1 | | | | Mode2 | | | Mode4 | |
|---|---|---|---|---|---|---|---|---|---|
| Robot | Rb1 | Rb2 | Rb3 | Rb4 | Rb1 | Rb3 | Rb4 | Rb1 | Rb3 |
| Energy-1 | 6690 | 6690 | 6690 | 6690 | 4127 | 4127 | 4127 | 1525 | 1525 |
| Energy-2 | 4726.9 | 3871.5 | 4303.4 | 5139.9 | 3424.2 | 2406.4 | 2427.4 | 1158.9 | 1220.5 |
| Saved energy | 1963.1 | 2818.5 | 2386.6 | 1550.1 | 702.8 | 1720.6 | 1699.6 | 366.1 | 304.5 |

It means that when robot $Rb4$ breaks down, the two reconfiguration functions $r_{1,3}$ and $r_{1,s}$ are possible to fire simultaneously.

The triggering conditions of minor reconfiguration functions can be computed previously. There are several possible state/event paths showing system behavior from the initial state to the objective state, at which 100 subassemblies are finished. We select a minimal path regarding to time for each TNCES-based model of the three configurations, to be denoted by $Path = \mathcal{Z}_1, \mathcal{Z}_2, ....\mathcal{Z}_n$, where energy-efficient operations are not included. That is to say all robots should stay in their working modes in this case although they should wait for a period of time before the next task comes. After that, based on the states on this path, the time when a minor reconfiguration function gets enabled and fires can be computed. For example, if an activated robot starts to wait at a particular state $\mathcal{Z}_i$, at which the system time is $\tau_1$. A search is performed along this minimal path at $\tau_1$. If it is found that at $\mathcal{Z}_j$, the robot works again, at which the system time is $\tau_2$. Then the time delay $\Delta\tau = \tau_2 - \tau_1$ between these two states is obtained. The round local reconfigurations for switching a robot between its working mode and energy-efficient mode takes two time units. Therefore, if the time delay is larger than two, *i.e.*, $\Delta\tau > 2$, a local reconfiguration can be applied to this robot. The system time for reconfiguring this robot from its working mode to its energy-efficient mode is $\tau_1$. The system time for reconfiguring this robot from its energy-efficient mode to its working mode is $\tau_2 - 1$.

The time of robots on their energy-efficient modes in minimal pathes is computed during the assembly of 100 subassemblies. They are shown in Table 6.1 together with the whole system uptime in each mode. Take Mode1 as an example. Assume that $Rb1$ consumes one energy unit per time unit in its working time but only consumes $30\%$ energy unit per time unit in its energy-efficient mode. In $Mode1$, if there is no minor reconfiguration applied to $Rb1$ for saving energy, it will consume 6990 energy units. However, it only consumes $6990 - 3233 + 30\% \times 3233 = 4726.9$ energy units in Mode1 if minor reconfigurations are applied when it is idle. In the same way, the energy saved by the robots during this simulation is shown in Table. 6.2, where the third row shows the energy consumption of each robot if no minor reconfigurations are applied, the fourth row shows the energy consumption of each robot when minor reconfigurations are applied, and the last row shows the saved energy of each robot during this process.

## 6.5 Summary

A reconfigurable and energy-efficient manufacturing system (REMS) is a typical reconfigurable discrete event control system (RDECS). It allows two kinds of dynamic system reconfigurations: local and global reconfigurations. The former are applied to save energy for components, whereas the latter are applied to change system configurations according to changed inner/outer execution environments. Meanwhile, normal events should be conditionally allowed to occur simultaneously with these system reconfigurations, such that the system can reconfigure smoothly and safely. In order to easily model conditioned concurrence of reconfiguration events and normal events and represent all interesting system behavior, this chapter extends R-TNCESs. Original reconfiguration functions are newly assigned with action ranges and concurrent decision functions. Accordingly, the dynamics of R-TNCESs is updated. After that, a TNCES-based implementation method for the proposed extended R-TNCESs is developed such that automatic model checking can be applied. The verified properties include functional, temporal, and energy properties that are specified by CTL, eCTL, or TCTL. An automatic assembly system is used to illustrate the whole work.

# Chapter 7 Conclusion

This dissertation deals with formal modeling and verification of dynamic reconfigurable discrete event control systems (RDECSs) based on Net Condition/Event Systems (NCESs) or Timed Net Condition/Event Systems (TNCESs). As a conclusion, this chapter summarizes contributions of this dissertation, discusses open problems of the current work, and introduces prospective future work on RDECSs.

## 7.1 Contribution

Formal modeling is the first and a critical step of accurate analysis of complex control systems. Formal verification is an expected method to check the quality of service of control systems completely, since it is also able to prove whether the system is incorrect. The verification results can help designers and engineers to improve the original design scheme. These techniques are always applied before start-up of a system. Therefore, many latent design deficiency can be discovered and solved earlier. In theory, man-made modern technological systems can be studied by considering them as discrete event systems (DESs). Considering the potential benefits of NCESs/TNCESs in modeling and verification of reconfigurable systems, all works described in this dissertation are carried out based on them.

In the beginning, this dissertation models a reconfigurable control system by an NCES. A reconfiguration scenario is defined by any addition-removal-update of places, transitions, or just modification of the initial markings. Three nested external formal modules are developed to cope with these reconfigurations. The first module deals with places, the second deals with transitions, and the third deals with initial markings. In order to guarantee safe behavior of this reconfigurable architecture, computation tree logic (CTL) based functional properties are checked by the software SESA.

From the first work described in Chapter 3, it is noticed that the modeling methods for RDECSs by using NCESs directly will increase the verification complexity sharply. This can be explained as follows. The original system model itself as well as the external control module generate a large state space. After they are synthesized, the state-space amplifies exponentially. As a result, we turn to study a direct and compact modeling method. A new formalism Reconfigurable Timed Net Condistion/Event System (R-TNCES) for modeling and verification of RDECSs is developed. Compared with the previous studies on formal methods for RDECSs, the functional and temporal specifications are optimized, and more forms of reconfiguration scenarios are covered such as the addition/removal of control component modules and the modifications of condition/event signals among them. Especially, a reconfiguration function in an R-TNCES has not only a structure modification instruction to dispose the structure reconfiguration but also a state processing function to assure the coherence of system states before and after any implementation of reconfiguration scenarios. Therefore, an R-TNCES is such a formalism that guarantees the correctness of an RDECS from the viewpoint of the model.

In order to control the verification complexity of R-TNCESs, a layer-by-layer verification method for a special type of R-TNCESs is proposed by using the model-checker SESA. The similarity among TNCESs in the behavior module is considered, which helps to simplify the verification process. It is proved that if the external environment of the unchanged parts is not changed by reconfigurations, their repetitive verification can be avoided. This solution controls the complexity of model-checking of R-TNCESs. A benchmark production system FESTO MPS is taken as a whole running example in this work. It shows that an R-TNCES is a convenient formalism for modeling and analyzing RDECSs.

The consistency is one of the most critical problems of a dynamic reconfigurable system. To this end, this dissertation proposes a novel virtual coordinator for a distributed reconfigurable discrete event control system (DRDECS) to deal with the reconfigurable coordination of a group set of R-TNCES-based reconfigurable discrete event control subsystems. Concurrent reconfiguration requirements are well solved by judgement matrices with an optimal coordination solution, while the amount of exchanged messages between subsystems and the coordinator is well-controlled.

The concurrence of reconfiguration events and normal events is a natural dynamic property of RDECSs, which may cause faults if they are not well-controlled. In order to easily model conditioned concurrence of reconfiguration events and normal events and to represent all interesting system behavior, this dissertation further extends R-TNCESs. Original reconfiguration functions are newly assigned with action ranges and concurrent decision functions. Accordingly, the dynamics of R-TNCESs is updated. Recent research work shows that reconfigurable control technologies can be applied to save energy of manufacturing systems by actively switching machines between their working modes and energy-efficient modes. A reconfigurable and energy-efficient vehicle assembly system is modeled and analyzed by an extended R-TNCES.

## 7.2 Discussion and Future Works

This dissertation studies some aspects of RDECSs. However, all current work is only a tip of the iceberg. Many problems are identified little by little but not solved by the current work.

### 7.2.1 Discussion

The current work copes with RDECSs on the system level, where a dynamic reconfiguration process is abstracted as one single discrete event. This reconfiguration event is set to be with the highest firing priority and its firing is assumed to be instantaneous. Obviously, the detailed dynamic reconfiguration processes are not well modeled, let alone the analysis and control. In addition, both in R-TNCESs and extended R-TNCESs, the structure modification instruction is not studied intensively, since the current work assumes that the reconfigurable controller knows how to implement it. Furthermore, the state correlation function is with a similar case that all current work is carried out by assuming that there is an available state correlation function.

Although the R-TNCES formalism is extended in Chapter 6, there are still lots of problems waiting to be solved. As described in Chapter 6, a reconfiguration function has been assigned with an action range and a concurrent decision function. It likely that detailed system behavior can be modeled and further analyzed. However, questions such as how action ranges are defined, implementation orders of structure modification instructions of multiple enabled reconfiguration events, and how the concurrent decision functions work in the cases that action ranges of multiple enabled reconfiguration events have intersections, are not well answered yet.

A special formal verification software for R-TNCESs is lacking. The formal verification of R-TNCESs and extended R-TNCESs is realized with the help of the software SESA. However, this process is semi-automatic. For any reconfiguration scenario, we should feed the new obtained configuration into the software again. Obviously, SESA cannot offer the formal verification of detailed dynamic reconfiguration processes based on R-TNCESs. In addition, SESA does not offer a visual interface. Therefore, it is not convenient and all the system models have to be drawn by hand.

## 7.3 Future Work

So far, to the best of our knowledge, research works concerning dynamic reconfiguration processes are limited. In [107], Gierds et al. propose a reconfigurable control approach to adjust the communication among some given services such that a certain behavioral property holds in the composed system. However, they do not consider the correctness of the adaptation phase. Narges et. at develop a supervisory controller to guide the behavior of a software system during adaption [108]. The possible concurrence of normal events and reconfiguration events is ignored and the temporal properties of reconfiguration events are out of consideration. In addition, they only concern the structure modification of reconfigurable systems.

Considering problems identified by the current work and the lacking of efficient theoretical results on dynamic reconfiguration processes of RDECSs, we plan to solve following problems step by step: 1) Enrich the R-TNCES formalism; 2) Study the stability of dynamic RDECSs based on R-TNCESs; 3) Investigate optimal verification methods for R-TNCESs by considering net structure properties; 4) Develop a special simulation and model checking software for R-TNCESs. Besides, another axis of research could be the close integration of the proposed methodology with IEC 61499 for implementation and possibly IEC 61850 [109] for the application in distributed energy systems.

# Reference

[1] KHALGUI M, MOSBAHI O, ZHANG J, et al. Feasible Dynamic Reconfigurations of Petri Nets-Application to a Production System.[C] // ICSOFT (2). 2011 : 105 – 110.

[2] KHALGUI M, MOSBAHI O, LI Z, et al. Reconfiguration of distributed embedded-control systems[J]. Mechatronics, IEEE/ASME Transactions on, 2011, 16(4) : 684 – 694.

[3] STEINBERG M. Historical overview of research in reconfigurable flight control[J]. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 2005, 219(4) : 263 – 275.

[4] ZHANG Y, JIANG J. Bibliographical review on reconfigurable fault-tolerant control systems[J]. Annual reviews in control, 2008, 32(2) : 229 – 252.

[5] VANDER VELDE W E. Control system reconfiguration[J], 1984.

[6] CHANDLER P R. Self-repairing flight control system reliability and maintainability program executive overview[C] // Proc. Nat. Aero. & Electr. Conf. 1984 : 586 – 590.

[7] WEISS J S E J L, WILLSKY D P L A. Design issues for fault tolerant-restructurable aircraft control[C] // . 1985.

[8] RAUCH H E. Autonomous control reconfiguration[J]. Control Systems, IEEE, 1995, 15(6) : 37 – 48.

[9] BENITEZ-PÉREZ H, GARCIA-NOCETTI F. Reconfigurable Distributed Control[M]. [S.l.] : Springer, 2005.

[10] HAJIYEV C, CALISKAN F. Fault diagnosis and reconfiguration in flight control systems : Vol 2[M]. [S.l.] : Springer, 2003.

[11] BONDY J A, MURTY U S R. Graph theory with applications : Vol 290[M]. [S.l.] : Macmillan London, 1976.

[12] RESCHER N, URQUHART A. Temporal logic : Vol 220[M]. [S.l.] : Springer-Verlag New York, 1971.

[13] PETRI C A. Communicating with Automata[J]. Germany: PhD thesis, Technical University Darmstadt, 1962.

[14] TRAKHTENBROT B A, BARZDIN Y M. Finite automata[M]. [S.l.] : American Elsevier Publishing Company, 1973.

[15] NAGEL K, SCHRECKENBERG M. A cellular automaton model for freeway traffic[J]. Journal de physique I, 1992, 2(12) : 2221 – 2229.

[16] NILSSON N J. A mobile automaton: An application of artificial intelligence techniques[R]. [S.l.] :

DTIC Document, 1969.

[17] HENZINGER T A. The theory of hybrid automata[M]. [S.l.] : Springer, 2000.

[18] MARSAN M A, BALBO G, CONTE G, et al. Modelling with generalized stochastic Petri nets[M]. [S.l.] : John Wiley & Sons, Inc., 1994.

[19] RAMCHANDANI C. Analysis of asynchronous concurrent systems by timed Petri nets[J], 1974.

[20] JENSEN K. Coloured Petri nets: basic concepts, analysis methods and practical use : Vol 1[M]. [S.l.] : Springer Science & Business Media, 1997.

[21] HANISCH H-M, THIEME J, LUDER A, et al. Modeling of PLC behavior by means of timed net condition/event systems[C] // Emerging Technologies and Factory Automation Proceedings, 1997. ETFA'97., 1997 6th International Conference on. 1997 : 391 – 396.

[22] RAMADGE P J, WONHAM W M. The control of discrete event systems[J]. Proceedings of the IEEE, 1989, 77(1) : 81 – 98.

[23] LI J, DAI X, MENG Z, et al. Rapid design and reconfiguration of Petri net models for reconfigurable manufacturing cells with improved net rewriting systems and activity diagrams[J]. Computers & Industrial Engineering, 2009, 57(4) : 1431 – 1451.

[24] MURATA T. Petri nets: Properties, analysis and applications[J]. Proceedings of the IEEE, 1989, 77(4) : 541 – 580.

[25] LI Z, ZHOU M. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2004, 34(1) : 38 – 51.

[26] CHU F, XIE X-L. Deadlock analysis of Petri nets using siphons and mathematical programming[J]. Robotics and Automation, IEEE Transactions on, 1997, 13(6) : 793 – 804.

[27] COMMONER F G. Deadlocks in Petri-nets[M]. [S.l.] : Massachusetts Computer Assoc., Incorporated, 1972.

[28] TRICAS F, GARCIA-VALLES F, COLOM J M, et al. A Petri net structure-based deadlock prevention solution for sequential resource allocation systems[C] // Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. 2005 : 271 – 277.

[29] LO K, NG H, TRECAT J. Power systems fault diagnosis using Petri nets[C] // Generation, Transmission and Distribution, IEE Proceedings- : Vol 144. 1997 : 231 – 236.

[30] RAMÍREZ-TREVIÑO A, RUIZ-BELTRÁN E, RIVERA-RANGEL I, et al. Online fault diagnosis of discrete event systems. A Petri net-based approach[J]. Automation Science and Engineering, IEEE Transactions on, 2007, 4(1) : 31 – 39.

[31] RAUSCH M, HANISCH H-M. Net condition/event systems with multiple condition outputs[C] // Emerging Technologies and Factory Automation, 1995. ETFA'95, Proceedings., 1995 IN-

RIA/IEEE Symposium on : Vol 1. 1995 : 592 – 600.

[32] KHALGUI M, HANISCH H-M, GHARBI A. Model-checking for the functional safety of control component-based heterogeneous embedded systems[C] // Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on. 2009 : 1 – 10.

[33] KHALGUI M, MOSBAHI O, LI Z, et al. Reconfigurable multiagent embedded control systems: From modeling to implementation[J]. Computers, IEEE Transactions on, 2011, 60(4) : 538 – 551.

[34] VYATKIN V, HANISCH H-M. Verification of distributed control systems in intelligent manufacturing[J]. Journal of Intelligent Manufacturing, 2003, 14(1) : 123 – 136.

[35] VALK R. Self-modifying nets, a natural extension of Petri nets[G] // Automata, Languages and Programming. [S.l.] : Springer, 1978 : 464 – 476.

[36] GUAN S-U, LIM S-S. Modeling adaptable multimedia and self-modifying protocol execution[J]. Future Generation Computer Systems, 2004, 20(1) : 123 – 143.

[37] LLORENS M, OLIVER J. Structural and dynamic changes in concurrent systems: reconfigurable Petri nets[J]. Computers, IEEE Transactions on, 2004, 53(9) : 1147 – 1158.

[38] LI J, DAI X, MENG Z. Automatic reconfiguration of petri net controllers for reconfigurable manufacturing systems with an improved net rewriting system-based approach[J]. Automation Science and Engineering, IEEE Transactions on, 2009, 6(1) : 156 – 167.

[39] ALMEIDA E E, LUNTZ J E, TILBURY D M. Event-condition-action systems for reconfigurable logic control[J]. Automation Science and Engineering, IEEE Transactions on, 2007, 4(2) : 167 – 181.

[40] WU N, ZHOU M. Intelligent token Petri nets for modelling and control of reconfigurable automated manufacturing systems with dynamical changes[J]. Transactions of the Institute of Measurement and Control, 2009.

[41] SAMPATH R, DARABI H, BUY U, et al. Control reconfiguration of discrete event systems with dynamic control specifications[J]. Automation Science and Engineering, IEEE Transactions on, 2008, 5(1) : 84 – 100.

[42] DUMITRACHE I, CARAMIHAI S, STANESCU A. Intelligent agent-based control systems in manufacturing[C] // Intelligent Control, 2000. Proceedings of the 2000 IEEE International Symposium on. 2000 : 369 – 374.

[43] OHASHI K, SHIN K G. Model-based control for reconfigurable manufacturing systems[C] // Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on : Vol 1. 2001 : 553 – 558.

[44] KALITA D, KHARGONEKAR P P. Formal verification for analysis and design of logic controllers for reconfigurable machining systems[J]. Robotics and Automation, IEEE Transactions on, 2002,

18(4) : 463 – 474.

[45] LIU J, DARABI H. Control reconfiguration of discrete event systems controllers with partial observation[J]. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2004, 34(6) : 2262 – 2272.

[46] ASPERTI A, BUSI N. Mobile petri nets[J], 1996.

[47] VAN BRUSSEL H, WYNS J, VALCKENAERS P, et al. Reference architecture for holonic manufacturing systems: PROSA[J]. Computers in industry, 1998, 37(3) : 255 – 274.

[48] VALCKENAERS P, VAN BRUSSEL H, BONGAERTS L, et al. Holonic manufacturing systems[J]. Integrated Computer-Aided Engineering, 1997, 4(3) : 191 – 201.

[49] CATALÁN C, SERNA F, BLESA A, et al. Communication types for manufacturing systems. A proposal to Distributed Control System based on IEC 61499[C] // Automation Science and Engineering (CASE), 2011 IEEE Conference on. 2011 : 767 – 772.

[50] NOVÁK P, ROLLO M, HODÍK J, et al. Communication security in multi-agent systems[G] // Multi-agent systems and applications III. [S.l.] : Springer, 2003 : 454 – 463.

[51] BERNA-KOES M, NOURBAKHSH I, SYCARA K. Communication efficiency in multi-agent systems[C] // Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on : Vol 3. 2004 : 2129 – 2134.

[52] SIMS M, CORKILL D, LESSER V. Automated organization design for multi-agent systems[J]. Autonomous Agents and Multi-Agent Systems, 2008, 16(2) : 151 – 185.

[53] MAILLER R, LESSER V. A Cooperative mediation-based protocol for dynamic distributed resource allocation[J]. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2006, 36(1) : 80 – 91.

[54] MAILLER R, LESSER V. A mediation based protocol for distributed constraint satisfaction[C] // The fourth international workshop on distributed constraint reasoning. 2003 : 49 – 58.

[55] CORKILL D D. Turn off your radios! environmental monitoring using power-constrained sensor agents[J], 2007.

[56] KHALGUI M, HANISCH H-M. Automatic NCES-based specification and SESA-based verification of feasible control components in benchmark production systems[J]. International Journal of Modelling, Identification and Control, 2011, 12(3) : 223 – 243.

[57] MCMILLAN K L. Symbolic model checking[M]. [S.l.] : Springer, 1993.

[58] CLARKE E M, GRUMBERG O, PELED D. Model checking[M]. [S.l.] : MIT press, 1999.

[59] BOZGA M, DAWS C, MALER O, et al. Kronos: A model-checking tool for real-time systems[C] // Formal Techniques in Real-Time and Fault-Tolerant Systems. 1998 : 298 – 302.

[60] BEHRMANN G, DAVID A, LARSEN K G. A tutorial on uppaal[G] // Formal methods for the

design of real-time systems. [S.l.] : Springer, 2004 : 200 – 236.

[61] HENZINGER T A, HO P-H, WONG-TOI H. HyTech: A model checker for hybrid systems[C] // Computer aided verification. 1997 : 460 – 463.

[62] STARKE P H, ROCH S. Analysing signal-net systems[M]. [S.l.] : Professoren des Inst. für Informatik, 2002.

[63] http://homepages.engineering.auckland.ac.nz/ vyatkin/tools/modelchekers.html[K]. .

[64] SILVA B I, RICHESON K, KROGH B, et al. Modeling and verifying hybrid dynamic systems using CheckMate[C] // Proceedings of 4th International Conference on Automation of Mixed Processes. 2000 : 323 – 328.

[65] ASARIN E, DANG T, MALER O. The d/dt tool for verification of hybrid systems[C] // Computer Aided Verification. 2002 : 365 – 370.

[66] MITCHELL I M. A toolbox of level set methods[J]. Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, http://www. cs. ubc. ca/˜ mitchell/ToolboxLS/toolboxLS. pdf, Tech. Rep. TR-2004-09, 2004.

[67] ALUR R, COURCOUBETIS C, DILL D. Model-checking for real-time systems[C] // Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e. 1990 : 414 – 425.

[68] http://www2.informatik.hu-berlin.de/ starke/ina.html[K]. .

[69] REYNOLDS M. An axiomatization of full computation tree logic[J]. The Journal of Symbolic Logic, 2001, 66(03) : 1011 – 1057.

[70] ROCH S. Extended computation tree logic[C] // Workshop Concurrency, Speci & Programming, number 140 in Informatik-Bericht. 2000.

[71] VYATKIN V. Modelling and Verification of Discrete Control Systems[J], .

[72] VYATKIN V, HANISCH H-M, PANG C, et al. Closed-loop modeling in future automation system engineering and validation[J]. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2009, 39(1) : 17 – 28.

[73] VYATKIN V, HANISCH H-M. Formal modeling and verification in the software engineering framework of IEC 61499: a way to self-verifying systems[C] // Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on : Vol 2. 2001 : 113 – 118.

[74] KHALGUI M, HANISCH H-M. Automatic specification of feasible Control Tasks in Benchmark Production Systems[C] // Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on. 2008 : 789 – 798.

[75] KOLLURU R, VALAVANIS K P, SMITH S, et al. Design fundamentals of a reconfigurable robotic

gripper system[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2000, 30(2) : 181 – 187.

[76] CHEN Y, LI Z. Design of a maximally permissive liveness-enforcing supervisor with a compressed supervisory structure for flexible manufacturing systems[J]. Automatica, 2011, 47(5) : 1028 – 1034.

[77] CHEN Y, LI Z, KHALGUI M, et al. Design of a maximally permissive liveness-enforcing Petri net supervisor for flexible manufacturing systems[J]. Automation Science and Engineering, IEEE Transactions on, 2011, 8(2) : 374 – 393.

[78] CHEN Y, LI Z, ZHOU M. Behaviorally optimal and structurally simple liveness-enforcing supervisors of flexible manufacturing systems[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2012, 42(3) : 615 – 629.

[79] HAN S, YOUN H Y. Modeling and analysis of time-critical context-aware service using extended interval timed colored Petri nets[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2012, 42(3) : 630 – 640.

[80] GEHIN A-L, STAROSWIECKI M. Reconfiguration analysis using generic component models[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2008, 38(3) : 575 – 583.

[81] GHARBI A, KHALGUI M, ZHANG J, et al. Agent-based Fault Management of Embedded Control Systems.[C] // ICSOFT (2). 2011 : 277 – 280.

[82] ALUR R, YANNAKAKIS M. Model checking of hierarchical state machines[J]. ACM SIGSOFT Software Engineering Notes, 1998, 23(6) : 175 – 188.

[83] PARISINI T, SACONE S. Fault diagnosis and controller re-configuration: an hybrid approach[C] // Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings. 1998 : 163 – 168.

[84] KOREN Y, HEISEL U, JOVANE F, et al. Reconfigurable manufacturing systems[J]. CIRP Annals-Manufacturing Technology, 1999, 48(2) : 527 – 540.

[85] YASUDA G. A distributed autonomous control architecture for synchronization and coordination of multiple robot systems[C] // SICE Annual Conference (SICE), 2012 Proceedings of. 2012 : 1864 – 1869.

[86] PROENÇA J, CLARKE D, DE VINK E, et al. Dreams: a framework for distributed synchronous coordination[C] // Proceedings of the 27th Annual ACM Symposium on Applied Computing. 2012 : 1510 – 1515.

[87] ZHANG J, KHALGUI M, LI Z, et al. R-TNCES: A novel formalism for reconfigurable discrete

event control systems[J]. Systems, Man, and Cybernetics: Systems, IEEE Transactions on, 2013, 43(4) : 757 – 772.

[88] YIGIT A S, ULSOY A G, ALLAHVERDI A. Optimizing modular product design for reconfigurable manufacturing[J]. Journal of Intelligent Manufacturing, 2002, 13(4) : 309 – 316.

[89] PITT J, MAMDANI A. Communication protocols in multi-agent systems: a development method and reference architecture[G] // Issues in agent communication. [S.l.] : Springer, 2000 : 160 – 177.

[90] CLARKE E M, EMERSON E A, SISTLA A P. Automatic verification of finite-state concurrent systems using temporal logic specifications[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1986, 8(2) : 244 – 263.

[91] LEITÃO P, ALVES J, MENDES J M, et al. Energy aware knowledge extraction from Petri nets supporting decision-making in service-oriented automation[C] // Industrial Electronics (ISIE), 2010 IEEE International Symposium on. 2010 : 3521 – 3526.

[92] KARNOUSKOS S, COLOMBO A W, LASTRA J L M, et al. Towards the energy efficient future factory[C] // Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on. 2009 : 367 – 371.

[93] BUNSE K, VODICKA M, SCHÖNSLEBEN P, et al. Integrating energy efficiency performance in production management–gap analysis between industrial needs and scientific literature[J]. Journal of Cleaner Production, 2011, 19(6) : 667 – 679.

[94] MECHS S, LAMPARTER S, MULLER J. On evaluation of alternative switching strategies for energy-efficient operation of modular factory automation systems[C] // Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on. 2012 : 1 – 8.

[95] MECHS S, MULLER J, LAMPARTER S, et al. Networked priced timed automata for energy-efficient factory automation[C] // American Control Conference (ACC), 2012. 2012 : 5310 – 5317.

[96] BI Z, WANG L. Optimization of machining processes from the perspective of energy consumption: a case study[J]. Journal of manufacturing systems, 2012, 31(4) : 420 – 428.

[97] ODA Y, KAWAMURA Y, FUJISHIMA M. Energy consumption reduction by machining process improvement[J]. Procedia CIRP, 2012, 4 : 120 – 124.

[98] CANNATA A, KARNOUSKOS S, TAISCH M. Energy efficiency driven process analysis and optimization in discrete manufacturing[C] // Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE. 2009 : 4449 – 4454.

[99] MOUZON G, YILDIRIM M B. A framework to minimise total energy consumption and total tardiness on a single machine[J]. International Journal of Sustainable Engineering, 2008, 1(2) : 105 – 116.

[100] SHORIN D, ZIMMERMANN A. Model-based development of energy-efficient automation sys-

tems[C] // The 17th IEEE Real-Time and Embedded Technology and Applications Symposium. 2011.

[101] PARK C-W, KWON K-S, KIM W-B, et al. Energy consumption reduction technology in manufacturing—A selective review of policies, standards, and research[J]. International Journal of Precision Engineering and Manufacturing, 2009, 10(5) : 151 – 173.

[102] STOFFELS P, BOUSSAHEL W M, VIELHABER M, et al. Energy engineering in the virtual factory[C] // Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on. 2013 : 1 – 6.

[103] WU N, ZHOU M, LI Z. Resource-oriented Petri net for deadlock avoidance in flexible assembly systems[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2008, 38(1) : 56 – 69.

[104] LI Z-W, ZHOU M. Two-stage method for synthesizing liveness-enforcing supervisors for flexible manufacturing systems using Petri nets[J]. Industrial Informatics, IEEE Transactions on, 2006, 2(4) : 313 – 325.

[105] LI Z W, HU H S, WANG A R. Design of liveness-enforcing supervisors for flexible manufacturing systems using Petri nets[J]. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2007, 37(4) : 517 – 526.

[106] LI Z, ZHOU M. Control of elementary and dependent siphons in Petri nets and their application[J]. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 2008, 38(1) : 133 – 148.

[107] GIERDS C, MOOIJ A J, WOLF K. Reducing adapter synthesis to controller synthesis[J]. Services Computing, IEEE Transactions on, 2012, 5(1) : 72 – 85.

[108] KHAKPOUR N, ARBAB F, RUTTEN E. Supervisory Controller Synthesis for Safe Software Adaptation[C] // Discrete Event Systems : Vol 12. 2014 : 39 – 45.

[109] MACKIEWICZ R. Overview of IEC 61850 and Benefits[C] // Power Systems Conference and Exposition, 2006. PSCE'06. 2006 IEEE PES. 2006 : 623 – 630.

[110] MEHRABI M G, ULSOY A G, KOREN Y. Reconfigurable manufacturing systems: Key to future manufacturing[J]. Journal of intelligent Manufacturing, 2000, 11(4) : 403 – 419.

# Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisors Prof. Zhiwu Li, Prof. Georg Frey, and Prof. Mohamed Khalgui.

I thank Prof. Zhiwu Li for his thoughtful guidance, instructive advice, constant encouragement, and financial support. He patiently taught me to pay attention to details in doing research and writing papers. He zealously finds promotion chances for me. I could not have imagined having a better advisor and mentor for my Ph.D study. Without his recommendation, I could not get chances to know my other two supervisors.

Prof. Georg Frey is my supervisor in Saarland University. It is my lucky to have him as one of my supervisors. His fair and serious attitude to me and to other formal registered Ph.d students touched me, when I was just a visiting student in his lab in the beginning. His speeches are always succinct and explicitly, which never hidden his enthusiasm. I appreciate him for his professional guidance on my research work and warm help on my life in Germany.

I would like to specially express my gratitude to Prof. Mohamed Khalgui, who has done a great contribution to every piece of my research work during the passed 5 years. He is always strict with my research work. I can hardly make progress in doing research without his guidance, criticism, and help.

Besides my supervisors, I would like to specially express my sincere appreciation to Prof. Kamel Barkaoui, Prof. Hans-Michael Hanisch, Prof.Olfa Mosbahi, Prof. Alessandro Giua, Prof. Naiqi Wu, Prof. Jianyuan Jia, Prof. Yuanyin Qiu, Prof. Helmut Seidel, Prof. Fernando Tricas García, Prof. Zhi Li, and Prof. Kai Cai for their help and concern.

I thank my fellow lab mates in Xidian University. It is my pleasure to conduct research with them for more than 3 years. Particularly among them are Dr. Ding Liu, Dr. Yifan Hou, Dr. Meng Qin, Dr. Yufeng Chen, Dr. Anrong Wang, Dr. Jinwei Guo, Dr. Gaiyun Liu, Dr. Chunfu Zhong, Dr. Hesuan Hu, Dr. Liang Hong, Ms. Na Li, Dr. Xiaoliang Chen, Dr. Zhongyuan Jiang, M.Sc. Xi Wang, Dr. Xiuyan Zhang, Ms. Yin Tong, Ms. Miao Liu, Mr. Ziyue Ma and Mr. Zhou He.

My sincere thanks also goes to all my colleagues and friends in Saarland University. Particularly among them are Ms. Gisela Auert-Kempka, Mr. Manfred Rachor, Dr.-Ing. Felix Felgner, Dipl.-Ing. Philipp Bauer, M.Sc. Fethi Belkhir, M.Sc. Wassim Boussahel, Dipl.-Ing. Lukas Exel, M.Sc. Mohammed Hijjo, Dipl.-Ing. Josef Meiers, Dipl.-Ing. Marco Nesarajah, and M.Sc. Christian Siegwart. I spent a pleasant time with them in Saarbrücken in Germany. They are full of zeal for helping me all the time. I will never forget the wonderful period with them.

I would like also express my gratitude to my Chinese friends in Germany. They are Mr. Jianguo Zhao, Dr. Huacheng Qiu, M.Sc. Yuchun Xing, Ms. Jing Niu, Mr. Guanghou Zhou, Mr. Xingtong Jiang, Ms. Xiangping Li, Dr. Dechen Chen, Dr. Ran Duan, Dr. Ye Liang, Mr. Yu Meng, Mr. Jingyu Yang, Mr. Jianxing Chen, Mr. Han Du, Ms. Xue Li, and Dr. Qiang Fu. I am grateful that God arranged them around me.

I am deeply grateful to my beloved parents and my sisters. No word can properly express how grateful I am to them. Their prayer for me was what sustained me thus far. At the end I would like to express appreciation to my beloved husband Shuo Tang, who is always my support. I will use my entire lifetime to love and protect him.