TAMPERE UNIVERSITY OF TECHNOLOGY

WAHEED AHMAD

FORMAL MODELLING OF COMPLEX EVENT PROCESSING AND ITS APPLICATION TO A MANUFACTURING LINE

Master of Science Thesis

Examiner: Professor Jose L. Martinez Lastra
Examiner and topic approved in the Automation, Mechanical and Materials Engineering Faculty Council on 07.12.2011

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Machine Automation
AHMAD, WAHEED: Formal Modelling of Complex Event Processing and its Application to a Manufacturing Line
Master of Science Thesis, 81 pages
January 2012
Major subject: Factory Automation
Examiner: Professor Jose L. Martinez Lastra
Keywords: Complex Event Processing, Event Processing Language, Formal Methods, Timed Net Condition Event System

Identifying the significant and most needed information in huge enterprises at the right time not only helps in decision making, but also plays an important role in overall performance and profit making of enterprises. Complex Event Processing (CEP) is a developing method of processing different events from multiple sources and filtering them to produce complex events.

This thesis provides a methodology to model CEP using Timed Net Condition Event System (TNCES), a Petri Nets derived formalism. Petri Nets is a graphical, mathematical modelling language used to analyze and describe discrete-event dynamic systems. The biggest advantage of representing CEP in TNCES is that it opens paths to the validation of the events filtering and decision making in different level of enterprise.

PREFACE

This thesis not only describes the end of my Masters studies, but also represents small journey in which I came across many new experiences and lessons. In this journey, I am especially grateful to my teachers in Tampere University of Technology who passed every possible help, knowledge and support to me.

I would also like to extend my profound gratitude to my supervisor Prof. Jose L. Martinez Lastra for giving me an opportunity to be a part of his research group FAST. Other than working on my thesis, my experience in FAST helped me a lot in grooming myself personally.

I am also highly indebted to my co-supervisor Dr. Andrei Lobov for his guidance, support, mentoring, advices, and patience during my thesis work.

My heartfelt thanks go to all my friends in Finland and at home for my moral support and care. My thanks are due to Ahmad Tariq, Tausif Babar and Faraz Amjad for guiding me and supporting me at each step of my studies and working life. I am also highly grateful to Muhammad Ahsan, Abdur Rehman, Ataul Ghalib, Faraz Ahmed, Azaz Ahmad, Masoom Ahmad, Waqas Ahmed Malik, Bin Zhang and Jorge Leal for making me feel Tampere is my home.

Most importantly, my deep-heart thanks to my family at this moment, for their unconditional love, support and guidance.

Waheed Ahmad
Tampere, January, 2012

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BOM	Bill of Materials
CEP	Complex Event Processing
CTL	Computational Tree Logic
EDA	Event Driven Architecture
EPL	Event Processing Language
ERP	Enterprise Resource Planning
IEEE	Institute of Electrical and Electronics Engineers
IL	Instruction List programming language
MES	Manufacturing Execution System
MOVIDA	Modeling and Formal Verification of Industrial Programming in Discrete Automation
PLC	Programmable Programming Logic
PN	Petri Nets
RFID	Radio Frequency Identification
(T)NCES	(Timed) Net/Condition Event System
UML	Unified Modeling Language
XML	eXtensible Markup Language

1. INTRODUCTION

This chapter gives an overview of the whole thesis topic. It contains background, research problem, objectives, hypothesis and contributions.

1.1. Background and Significance

From early 80's, manufacturing industries slowly started shifting towards distributed automation from centralized control approach (Ahmed Hambaba 1999). The major reasons are modernization, rapidly changing global competition, huge growth of technology and fast manufacturing times to name some. As a result manufacturing industries tend towards demand-driven more than plan-driven.

On one side, distributed intelligence in automation has provided advantages like robustness, better manageability, simplification and efficiency of highly complex systems. But, on the other side, one of the core issues which arose for management is availability of relevant information instantly in order to take decisions and manage process efficiently. But, this requires highly well-organized and efficient cross-layer communication which is able to provide relevant information to its destination in a very short time (Baoan Li and Minxing Li 2009).

This issue has given birth to a concept of Complex Event Processing (CEP) which includes taking in account many events generated in an organization from heterogeneous sources and then meaningful events are extracted out of them. Those meaningful events are processed in real time and forwarded to respective sections of the organization. In this way, process of acquiring information speeds up to a great extent (Heinz Roth et. al 2010). Using the concept of CEP, managers, engineers and stake holders can get required information in a very easy and simple manner. Other than getting information, instructions and update can also be taken utilizing CEP.

With the advancement of computer technologies, it is relatively easy now to capture events generated at different levels of enterprise with the help of very sensitive sensors and thus, providing faster response. Also, with the help of different technologies evolved in last decade, communication inside an industry has improved a lot. Using these technologies, it is easy to integrate different equipments and communication protocols in an enterprise in an Event Driven Architecture (EDA). Service Oriented Architecture (SOA) is one of paradigm which allows us to integrate different systems based on Services. (Curl, A., Fertalj, K 2009) (Ravier, Dominique 2010) (Tang Yongzhong 2009) (Jianfeng Qian, Jianwei Yin, Dongcai Shi, Jinxiang Dong 2008).

Business Process Modelling (BPM) is another option to integrate low level equipments on the factory floor. Business Process Execution Language (BPEL) or Web Ser-

vices Choreography Description Language (WS-CDL) can be also other possible selections to implement commercial process engines (workflow engines) in enterprise servers.

1.2. Research Problem

CEP is an emerging technology which is able to identify meaningful events out of event streams and creating complex events using correlation, abstraction and causality between events. But, in today's agile world, technologies need more usability and validation to ensure fulfilment of specified requirements. According to Mark R. Blackburn and Robert T. Busser (1998), the usability is one of the key features for adopting formal methods in industry. This thesis concentrates on application of formal methods on CEP and formal validation of systems.
Thus, research problem for this thesis can be states as follows:

"The development of methodology for complex event processing to represent event aggregation and filtering and validation of methodology using formal methods."

1.3. Research Description

1.3.1. Objectives

The main objectives of this thesis are formulated as:

O1: To develop methodology to model complex event processing using formal methods.
O2: To express system requirements using Computational Tree Logic (CTL) and perform validation based model-checking.

1.3.2. Hypothesis

The main hypothesis for the work documented in this thesis is that if a manufacturing line generating events is translated into modular typed representations, then it is possible to devise a methodology to construct a model to influence generation of complex events out of that line.

1.4. Contributions

This primary contribution of this thesis is a development of methodology for typed and modular modelling of complex event processing. This methodology can be applied on existing complex event processing concepts as well as newly introduced. The approach followed in this document also provides a novel means of validating a system.

1.5. Limitations of Scope

This thesis presents a methodology in modelling existing techniques and concepts of complex event processing in terms of Petri Nets. However, this thesis does not formulate any new language to process events in enterprise architecture.

1.6. Outline

The remaining chapters of this thesis are structured as follows. Chapter 2 presents the literature review which describes most notable and significant contributions in the field. Chapter 3 explains the proposed methodology for modelling. Chapter 4 discusses implementation on case study based on methodology and obtained results. Chapter summarizes whole thesis, explains contributions made and possible research trends.

2. STATE OF THE ART

This chapter provides a deep insight into previous research done. The chapter starts with introduction and explanation of Complex Event Processing and its application in Event Driven Architecture in Section 2.1. A comprehensive overview of Event Processing Language along with different examples is presented in Section 2.2. The Section 2.3 gives an introduction of Event Pattern Language and examples related to it.

A comprehensive overview of Formal Methods theory is presented in Section 2.4. The chapter ends with description of different applications of Formal Methods theory in factory automation.

2.1. Complex Event Processing in Event Driven Architecture

2.1.1. Introduction

Event is regarded as the meaningful change in the system (Y.H.Zhang, Q.Y.Dai and R.Y.Zhong 2009) and they occur very frequently in manufacturing system. Common examples are placing an order by customer by calling customer service, sending machine settings to workers, receiving of raw materials by warehouse and informing manager about completion of order. These all events tend to inform system about any change and expect response from system.

Events occurring in a distributed, heterogeneous databases and applications are linked together to form a so called event cloud (Adrian Paschke 2009). *Complex Event Processing* is to identify most meaningful events within whole event cloud to produce *complex events*, analyzing those complex events, processing them and taking any action in real time.

For this reason, complex event processing is an emerging field and is one of the biggest research areas in industries. As organizations are expanding in terms of quantity of production and range of products, amount of events produced are also increasing by a huge amount. As a result of high competition, companies need more agility and this demand ability to manage unexpected amount of real-time data about their shop floor, performance, supply chain information and enterprise information. Also, businesses should have ability to extract meaningful information from seemingly irrelevant bunch of data.

To cope with these challenges, businesses need a compact methodology to capture all information irrespective of its location of occurrence and analyzing them. With CEP, businesses can link their key performance indicators (KPIs) like revenue, profit, maintenance cost etc. with events.

To elaborate CEP more, consider two examples with same result i.e. cancelled customer order. In first example, customer tries to place an order through company's ERP. But, router is down and as a result, customer is unable to place an order. Customer calls customer care of the company but customer care is unaware of this problem. He will eventually place an order on competitor's ERP.

In second example, a product is dispatched from the company but unfortunately driver of the truck goes to the wrong airport which will result in the late delivery. The customer will cancel the order or will place the order to the competitor next time. (Alan Lundberg)

In both scenarios, customer care centre or marketing department of the company needs to be in contact with customer from the time of placing the order on ERP to the time where product reaches its final destination. To prevent all this mishap effectively, businesses need a real time architecture that can identify most needed activities or events e.g. throughput of router or location of driver. Then we can respond to them beforehand to avoid any mishap and CEP serves this purpose really well.

To explain how CEP works, consider mentioned examples of cancelled order. In first example, if router generates events representing its throughput, then IT department of the company can know about the failure of the router or it is about to fail, then they can reroute data via any other backup router and meanwhile they can try to fix the issue.

In second example, if location of driver is informed to the customer care of the company by producing events, then customer care can know about the mistake of driver in advance, and then they can inform customer in advance and can offer discount on delayed order delivery.

In order to combine simple events to produce complex events, there exists a need to establish *causality* between events and set of causal events are called posets (partially ordered set of events). (David C. Luckham and Brian Frasca 1998) (Kshemkalyani, A.D. 1997) (Scherl R. and Shafer G., 1998)

The process of sorting out interesting events is done by *filters* that take posets of events as input and filter meaningful events. Filtering of events depend on event pattern matching. (David C. Luckham and Brian Frasca 1998) (Urban, S.D., Biswas, I., Dietrich, S.W. 2006)

For the purpose of developing higher level events, *maps* also called aggregators are used. Workings of maps depend on pairs of input and output event patterns. Maps output the events in the output pattern, whenever subset of input event matches an input pattern. (David C. Luckham and Brian Frasca 1998)

The previous knowledge about event filtering and mapping can be reused to construct new CEP solutions with the help of *patterns*. There are many types of patterns which enable the design engineers to build efficient and robust CEP solutions by taking in account already built CEP models. (Adrian Paschke 2008).

Adrian Paschke (2009) has also designed a semantic CEP pattern language which is formal, mark up and human readable. The main vocabulary of the CEP pattern language

is defined by design ontology language (DOL) and semantics of the language is composed of three parts, explicit semantics, formal semantics and informal semantics.

2.1.2. CEP Applications in Event Driven Architecture

Very intense research has been done on event driven architecture (EDA) and their role in CEP. In a research paper by Shang Wengli, Duan Bin and Shi Haibo (2008), they have presented an event driven model for manufacturing execution system (MES) platform shown in Fig. 2.1. They have divided MES platform into production model, visualized modelling tool, real time message bus, heterogeneous data sources adaptor and MES application suites. Production model is further is divided into product model, factory model, event model and execution model. Product model is responsible for defining product, material and to build Bill of Materials (BOM). Factory model defines factory floor, equipments and devices. Event model is to define production event of manufacturing process and Execution model defines production and process rules and traceability of whole manufacturing process.

Based on this MES platform, response mechanism is also developed in which Event model defines the message in production as event. Execution model is dependent on Event model and after triggering of event, Execution model will be called to deal with the created event.

Event model is further divided into two subparts i.e. Event Configuration Module and Event Detection Module. Event configure module is responsible for configuration of event to production cell and Event detection module configures the event trigger condition and to build business logic to dispose created event. This paper presents an excellent idea of dividing whole MES platform into sub-modules but this paper does not address communication and message sequence between sub-modules so well.

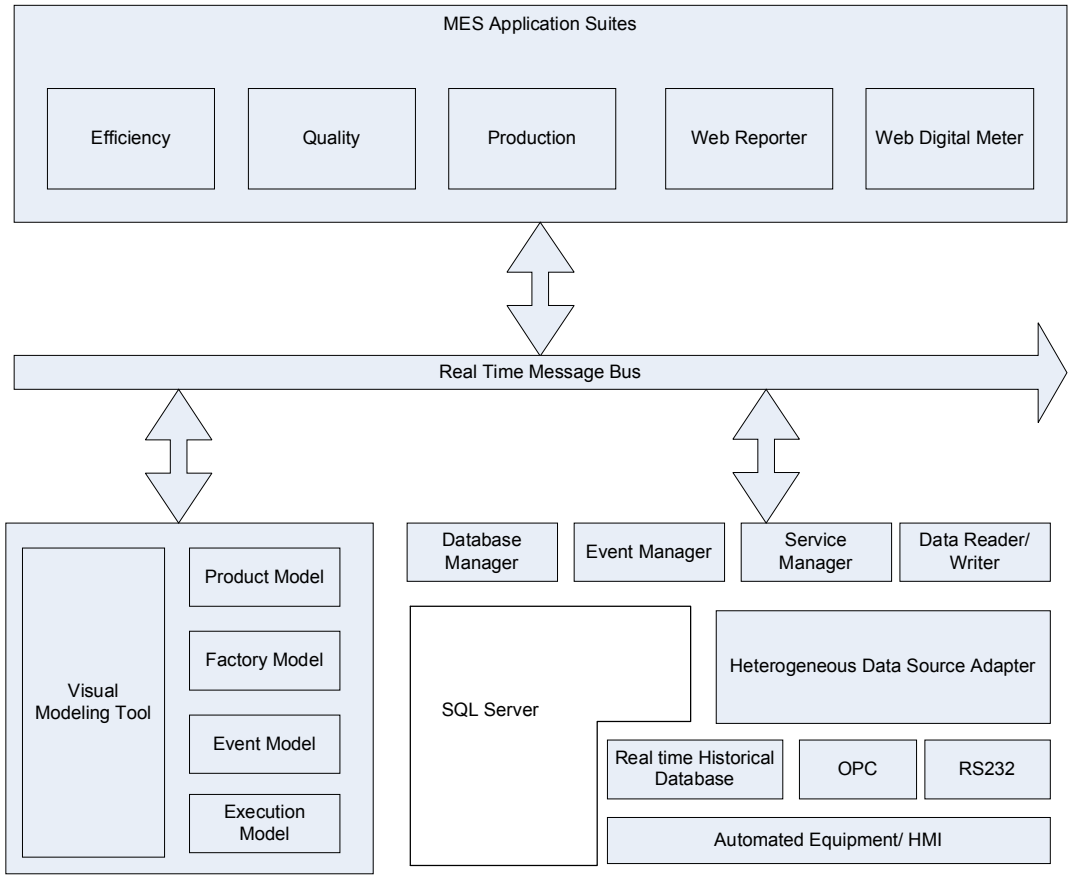


Figure 2-1 Event Driven Model for MES (Shang Wengli, Duan Bin, Shi Haibo 2008)

In another journal by Y.H.Zhang, Q.Y. Dai and R.H.Zhong (2009), idea of event-driven platform based on the MES is proposed. The proposed platform has loose coupling between its modelling components. The idea is that shop floor level will produce simple events while business level is more interested in statistical data. The proposed platform based on MES will receive the events from shop floor level and will inform the most meaningful data to enterprise level. The architecture is explained in more detail in Fig 2.2.

Overall architecture of this extensible platform consists of event analyzer, event configuration, event buffer zone, publisher/subscriber, customer platform and database. Database is used to store events and their information. Publish/subscribe (pub/sub) mechanism can be regarded as so called message broker and it distributes message efficiently. User Interface allows user to communicate with system. Event Configuration is the major part of the CEP and it registers and defines the event. In another words, it provides patterns for event detection. Event Buffer Zone (EBZ) is to store the events. As huge amount of events can be produced in seconds, EBZ provides buffer space in memory to store those events. Event analyzer is the most significant part of the whole system and its responsibilities include converting primitive events into complex events, link different parts of the system and connect all levels in the architecture.

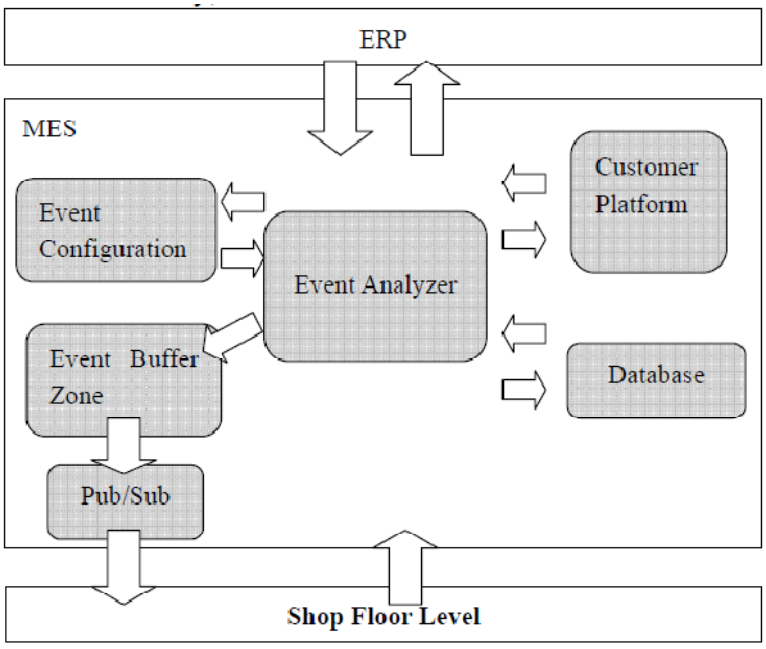


Figure 2-2 Architecture of Event Driven Model (Y.H.Zhang, Q.Y. Dai and R.H.Zhong 2009)

In another publication by Chuanzhen Zang, Yushun Fan (2007), architecture of complex event processing in enterprise information systems based on RFID is explained. Events produced by RFID, database and other enterprise information systems are processed by EPA (Event Processing Agent) before passing them to event bus. EPA filters duplicate events, resolve errors and match formats. In event bus, where process of aggregation into complex pattern take place according to specified rules.

Event meta-model is also devised in which intrinsic relation between different levels of event processing is shown. Also, the concept of event context is proposed. Event context is utilized in aggregating lower level primitive events into higher level complex events. Even context includes semantic space, workflow model and abstraction hierarchy. Semantic space is surrounded by two events named initiator and terminator. Initiator initiates the semantic space and occurrence of terminator terminates it.

Architecture of event server is also implemented which includes meta data, complex event modeller, complex event compiler, complex event detector and event receiving and publishing server. The complex event modeller is responsible for creating complex events, and calling complex event compiler to read and analyze the complex events. Meta data has all the definitions of the event patterns.

Complex event detector is the central part of the whole architecture. It implements all the semantics and optimization strategies. It is composed of complex event pattern cache, instance classification table and complex event classification table. First of all, event pattern cache extracts patterns from meta data. These complex event patterns help in building complex event classification table. Whenever any event instance will arrive to event receiving and publishing server, that instance will be classified into instance classification table. Complex event detector will take out the instance one by one ac-

cording to complex event classification table and process the pattern. Fig. 2.3 illustrates concept of architecture based on RFID in an EDA.

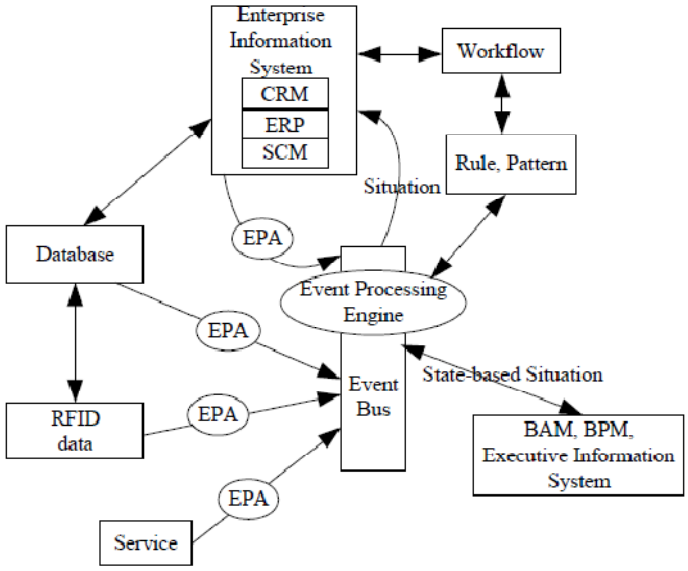


Figure 2-3 Architecture of CEP in Enterprise Information Systems (Chuanzhen Zang, Yushun Fan 2007)

Heinz Roth et. al (2010) have demonstrated an idea of Event Data Warehouse for CEP Applications to store and retrieve events efficiently. Whole architecture is divided into two parts i.e. Event Data Warehouse and Complex Event Processing. Events produced in shop floor will be fed into complex event processing part with the help of event adapters. The adapters collect events in push and pull scheme and pass them to Event Bus which will be forwarded to event processing models with the aid of sockets. After mapping, events will be forwarded to Event Data warehousing where event will be stored. From this storage, events can be retrieved with the help of any query language SQL and the retrieved event data can be utilized in analytical and operational purposes.

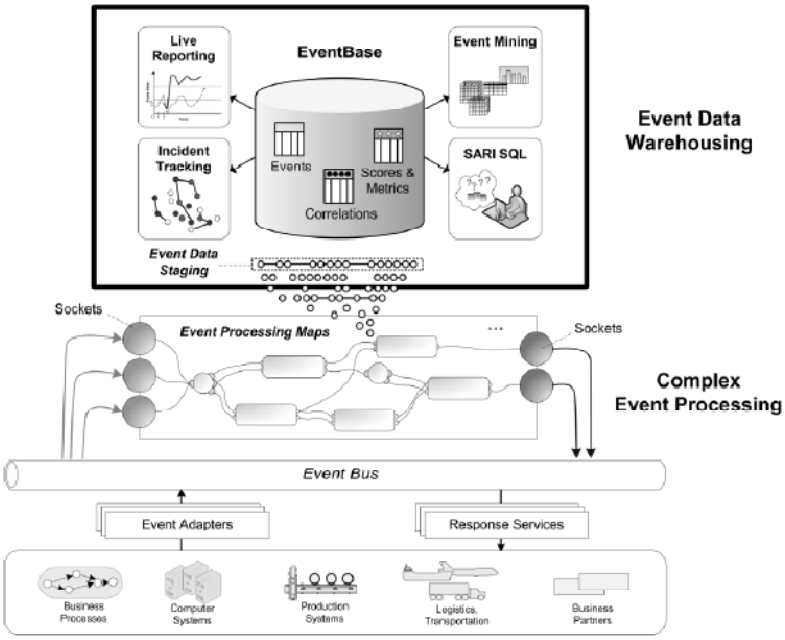


Figure 2-4 Event Warehouse for CEP Applications (Heinz Roth et. al 2010)

2.1.3. CEP in Distributed Systems

David C. Luckham and Brian Frasca (1998) introduced the concept of event hierarchy and flexible viewing in event abstraction hierarchies. Flexible view allows picking of only events which need 'attention', monitoring events and casual relationship between them at any stage of abstraction hierarchy and converting primitive events into complex events. Flexible view allows us to visualize events at any level of abstraction hierarchy. For example, any worker who is happy in viewing the workflow events and then suddenly some problems occur. Worker will change the view from higher level view to upper level view and determine the problem and its cause from relevant events.

Distributing whole complex system into smaller layers is called abstraction hierarchy and similarly we can develop event hierarchy which shows events at different layers. For example, we can divide control system of a conveyor line as middleware communication, point-to-point communication, workflow and product dispatch.

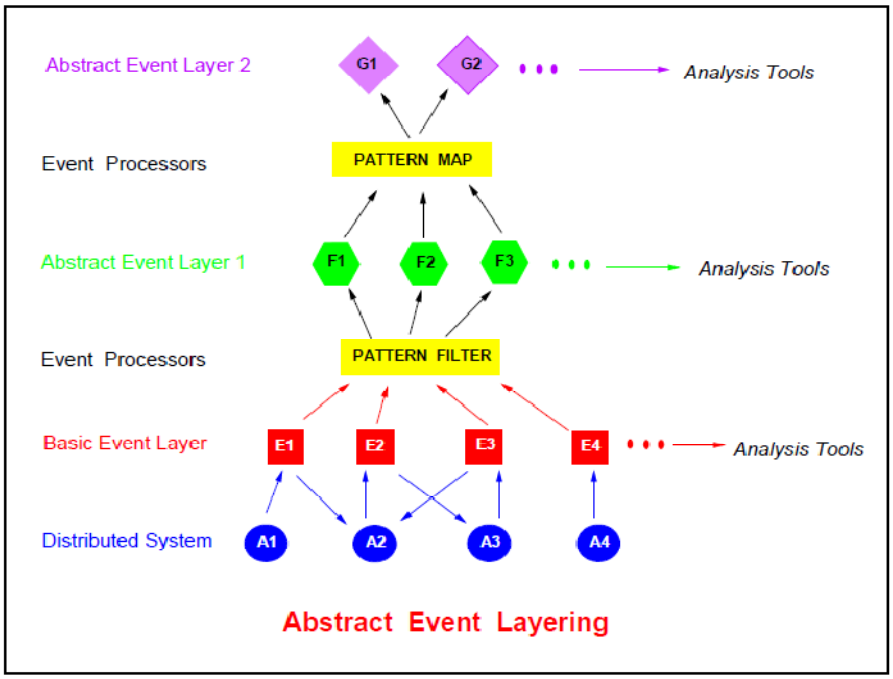


Figure 2-5 Concept of Event Hierarchy (David C. Luckham, Brian Frasca 1998)

2.2. Event Processing Language

Event Processing Language (EPL) is a SQL-like language which provides set of rules for processing events like filtering, correlating, applying constraints and aggregating. These set of rules helps to derive information from event streams and merging them together.

EPL provides the clauses like `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` and `ORDER BY`. Similar to table SQL, EPL provides the built-in views for placing the data. Moreover, EPL also provides the features like Patterns, Operators, Functions and Views.

A simple EPL query contains a `select` clause and a single event stream definition. But complex queries can be created by including different search conditions in `where` clause, or by elaborating `select` clause and so on. An example EPL query is given below in Code 2-1. (Yan Liu, Dong Wan 2010)

```
select select_list
from stream_def [as name] [, stream_def [as name]]
                [,...]
      [where search_conditions]
   [group by grouping_expression_list]
   [having grouping_search_conditions]
      [output output_specification]
   [order by order_by_expression_list]
```

Code 2-1 Select Clause Example

In the above query, *select* clause mentions which property to retrieve or select and *from* clause specifies the name of event stream. The *where* clause is used to specify search condition in order to filter output more and *Group by* clause divided the EPL output into groups. The *having* clause allow events defined by Group By clause to pass or reject them based on grouping search conditions. The *output* clause controls the rate of generation of output events. The *order by* clause order output events according to their properties.

2.2.1. EPL Syntax and Clauses

This section explains syntax of different EPL clauses and constructs (ESPER 2011).

Select Clause

It is required in all EPL statements and it is used to select event or event properties. The wildcard character * is used to select all the properties or specific list of properties can also be given.

Some special keywords like istream (input stream), rstream (remove stream) or ir-stream (input or remove stream) can also be used to input or remove stream. The syntax of where clause is given below.

```
select [istream | irstream|rstream] * | expres-
                sion_list...
```

Code 2-2: Select Clause Keywords

Choosing all event properties: select *

The syntax of selecting all events is,

```
select.* from stream_Def
```

Code 2-3 Selecting All Events

Choosing specific event properties

The syntax from choosing specific event properties is given below.

```
select event_property [, event_property] from stream_def
```

Code 2-4 Selecting Specific Properties

Expressions

The *select* clause may consist of one or more expressions,

```
select expression [, expression] [,  ...] from
                stream_def
```

Code 2-5 Multiple Expressions in Select Clause

Renaming event properties

By following syntax below, event properties can be renamed.

```
select [event_property | expression] as identifier[,...]
```

Code 2-6 Renaming Event Properties

Choosing events properties and events in a join

If we want to join multiple streams, then we can specify properties which are unique among the joined streams. In case, there is no unique property, then we need to use alias name of stream as a prefix of the property.

In the following example, two streams StockTick and News are joined, and named as ‘tick’ and ‘news’ respectively.

This example picks symbol value from StickTick event using ‘tick’ stream alias as prefix.

```
select tick.symbol from StockTick.win:Time(10) as tick,
                News.win:time(10) as news
```

Code 2-7 Joining Multiple Streams Example

The basic syntax of choosing events properties is,

```
select stream_name.* [as alias] from...
```

Choosing events properties and events from a pattern

If some expression uses pattern expressions, then those pattern expressions tag events with a tag name. Each tag name can be used as a property in the *selec*t clause and other clauses. Following example matches patterns whenever StockTick event is received within 30 seconds after start of statement. The symbol and price properties of the matching events are selected.

```
    select tick.symbol as symbol, tick.price as price
from pattern [every tick=StockTick where timer:within(30
                            sec)]
```

Code 2-8 Selecting Events from a Pattern Example

Filters

Filters are used to filter out events before entering data window. In the following example, only those Withdrawal events are selected which have amount value greater than 200.

```
select * from Withdrawal(amount>=200). win:length(5)
```

Code 2-9 Selecting Events using Filters Example

In the following figure, it can be seen that events W2, W4 and W5 cannot enter Length Window as their amount is less than 200.

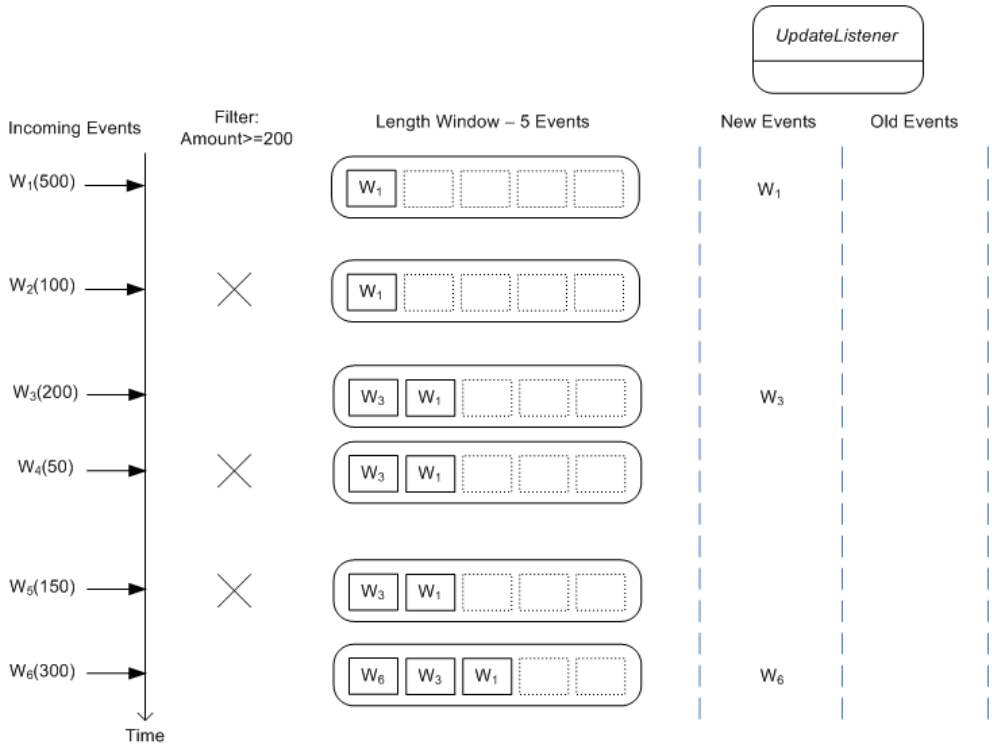


Figure 2-6 Statement with an event stream filter (http://esper.codehaus.org/)

Where Clause

In case of where clause, each event is allowed to enter length window, but only those events are updated to listeners as new events which are qualified by the where-clause. For example, in following only those events are updated to listeners having amount greater than 200.

```
select * from Withdrawal.win.length(5) where amount >=200
```

Code 2-10 Where Clause Example

In the figure below, W2, W4 and W5 are not posted to listeners as new events as their amount is less than 200.

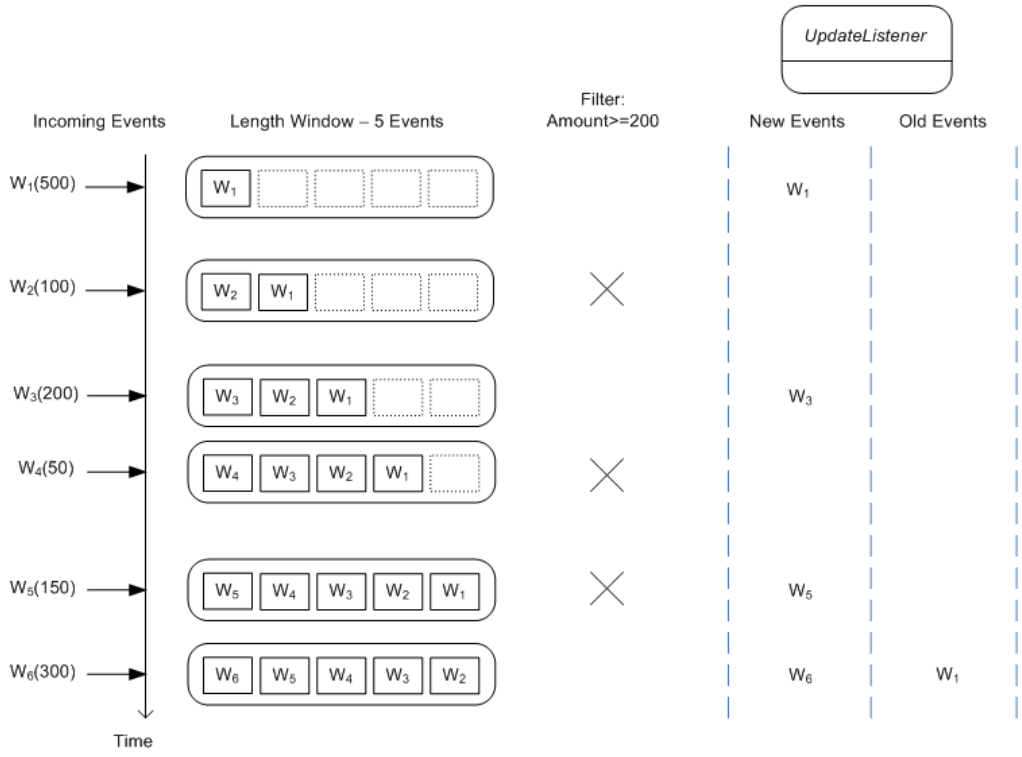


Figure 2-7 Statement with where-clause (http://esper.codehaus.org/)

Time Window

Time window is a window which can be extended to the specified time interval into the past. Following example selects the Withdrawal events in which amount per account is greater than 1000 in last 4 seconds.

```
select account, avg(amount)
from Withdrawal.win:time(4 sec)
group by account
having amount > 1000
```

Code 2-11 Time Window Example

In the following figure, event W1 occurs at t+4. After 4 seconds, event W1 leaves the window and it is posted as an old event to the listeners.

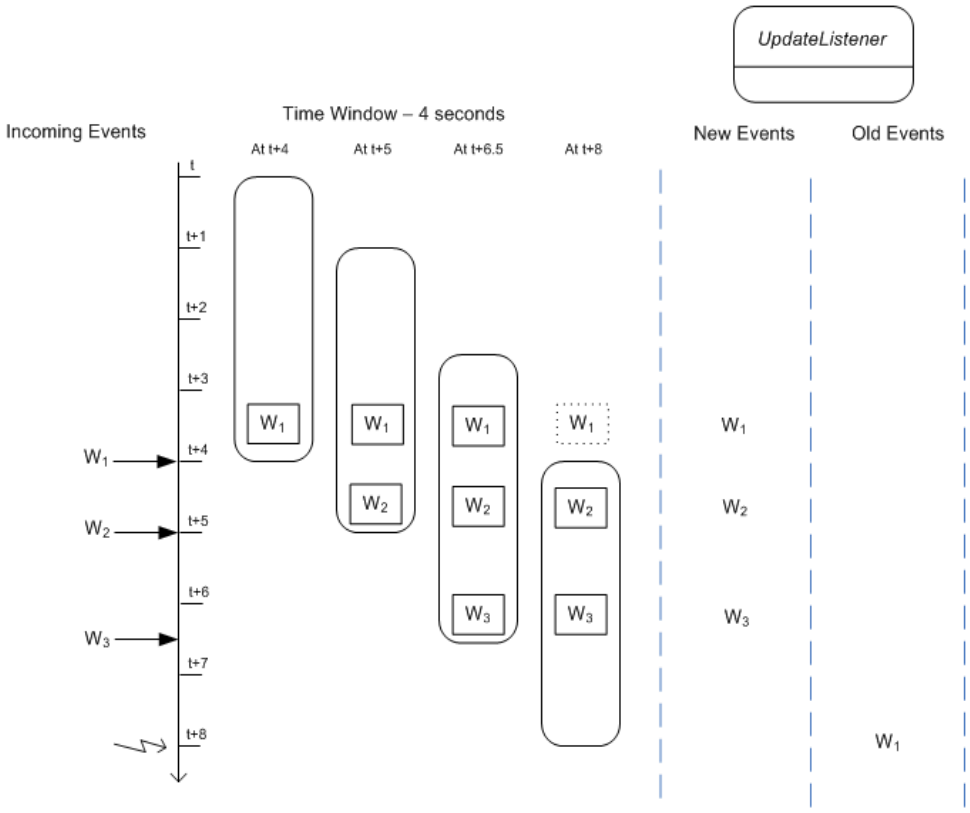


Figure 2-8: Statement with a time window (http://esper.codehaus.org/)

Time Batch

The time batch is used to buffer events and then to release them in a single update. In following EPL statement, Withdrawal events are buffered up to 4 seconds and released after 4 seconds.

```
select * from Withdrawal.event.win:time_batch(4 sec)
```

Code 2-12 Time Batch Example

In the following figure, W1 and W2 events are received at t+1 second and t+3 seconds respectively, but they are not updated to listeners. Then at t+4 seconds, engine processes the received events and update listeners about them.

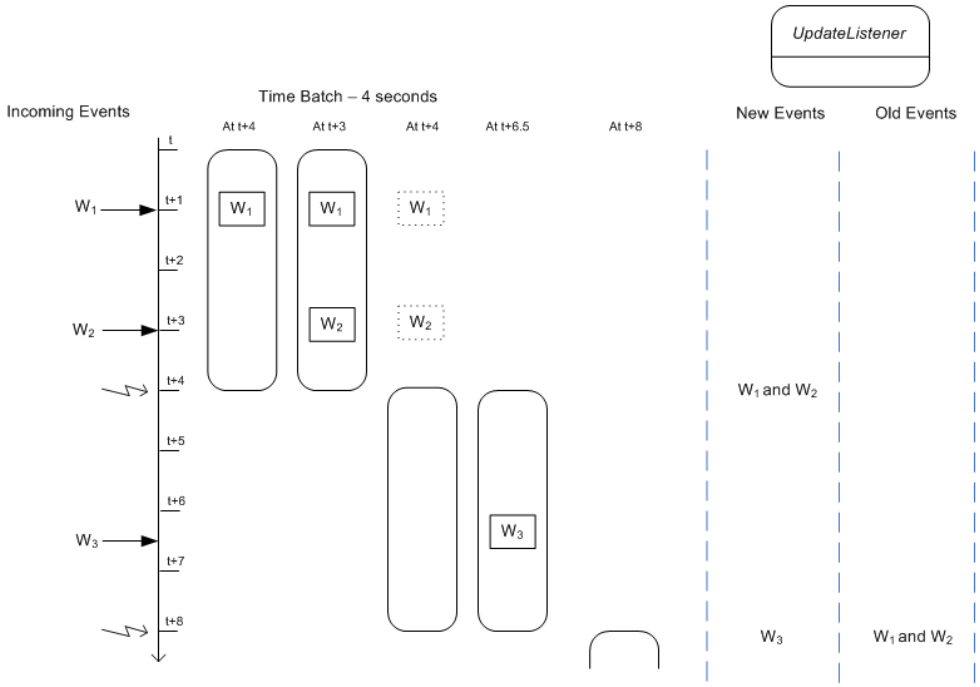


Figure 2-9 Statement with a time batch view (http://esper.codehaus.org/)

Aggregation and Grouping

Consider the following statement,

```
select count(*) as mycount from Withdrawal having
count(*)=2
```

Code 2-13 Aggregation and Grouping Example

Whenever 2 Withdrawal events are encountered, the engine posts an update event to listeners and updates value of 'mycount' property to 2.

From Clause

The *from* clause is required in each EPL statement. Multiple event streams and named windows can be mentioned in the *from* clause. The general syntax of the *from* clause is given below,

```
from stream_def [as name]  [, stream_def [as stream_name] ]
                           [, ...]
```

Code 2-14 From Clause Example

Specifying Filter Criteria

The criteria for filtering any stream of events is by placing it within parenthesis as shown below,

```
select * from RfidEvent (category= ”Perishable”)
```

Code 2-15 Specifying Filter Criteria Example

The following operators can also be used in order to enhance filtering process.

- equals =
- not equals !=
- comparison operators <, >, >= and <=

Filtering Ranges

Ranges can be of 4 types as given below,

- (low : high)- Open ranges not containing endpoints.
- [low : high]- Closed ranges containing endpoints. The equivalent keyword ‘between’ can be also be used.
- [low : high)- Half open ranges that contain lower endpoint but not the higher endpoint.
- (low : high]- Half closed ranges that contain the higher endpoint but not the lower endpoint.

Aggregate Functions

The aggregate functions are sum, avg, max, min, media, stddev, avedev. For example, following example calculates total price of all Stock tick events occurred in last 30 seconds.

```
select sum(price) from StockTickEvent.win:time(30 sec)
```

Code 2-16 Aggregate Function Example

Group-by Clause

It is an optional clause and it divided the output of an EPL statement into groups. The statement below, finds out total price per symbol from all Stock Tick events occurred in last 30 seconds.

```
select symbol, sum(price) from StockTickEvent.win:time(30
                    sec) group by symbol
```

Code 2-17 Group-by Clause Example

If group by clause result in null value, then null value becomes its own group and all null values are aggregated into the same group. To avoid this, where clause can also be used in which events not satisfying conditions are eliminated before any grouping is done.

Having Clause

With having clause, events defined by group-by clause are rejected or passed. For example, statement below calculates total price per symbol for events occurred in last 30 seconds only for those events symbols whose price is greater than 1000.

```
select symbol, sum(price)
from StockTickEvent.win:time (30 sec)
group by symbol
having sum(price) > 1000
```

Code 2-18 Having Clause Example

Interaction between Where, Group By and Having Clauses

To elaborate interaction between these 3 clauses, consider following statement.

```
select tickDataFeed, stddev(price)
from StockTickEvent(symbol='IBM').win:length(10)
where volume > 1000
group by tickDataFeed
having stddev(price) > 0.8
```

Code 2-19 Where, Group by and Having Clauses in Single Statement

In this example, events of only symbol IBM can enter length window over last 10 events and all other are discarded. The where clause excludes all those events whose volume is less than or equal to 1000. Then each tickDataFeed value generates one event and having clause let only those events for tickDataFeed groups whose standard deviation of price is greater than 0.8.

The Order By Clause

The *order by* clause is also optional and it is used to order output events according to their properties. The syntax of the *order by* statement is below,

```
order by expression [asc | desc] [, expression [asc |desc]
[, ...]
```

Code 2-20 Order By Clause

For example, the example below outputs batches of 5 or more events bys orting events according to their ascending price values and then according to ascending volume values.

```
select symbol from StockTickEvent.win:time(60 sec)
output every 5 events
order by price, volume
```

Code 2-21 Order By Example

2.2.2. EPL Operators

EPL operators with their description are given below (ESPER 2011).

Arithmetic Operators

The following table explains arithmetic operators available.

Table 2-1 Arithmetic Operators

Operator	Description
+,-	As unary operators, they represent positive or negative expression. As binary operator, they add or subtract.
*, /	Multiplication and division as binary operators.
%	Modulo binary operator

Logical and Comparison Operators

All logical and comparison with their description are outlined in table below.

Table 2-2 Logical and Comparison Operators

Operator	Description
NOT	Returns false if condition is true and returns true if condition is false.
OR	Returns true if either of the condition turns out to be true and false if all conditions are false.
AND	Returns true if all of the conditions are true and false if any of the conditions are false.
=,!=,<,>, <=,>=	Comparison

Concatenation Operators

The table below summarises concatenation operators of EPL.

Table 2-3 Concatenation Operator

Operator	Description
||	Concatenates character strings

Binary Operators

The binary operators used in EPL are described in following table.

Table 2-4 Binary Operators

Operator	Description
&	Bitwise or Conditional AND
|	Bitwise or Conditional OR
^	Bitwise XOR

Array Definition Operator

Arrays can be useful to pass to user-defined functions or to select any array data in a select clause. The curly braces { } are used to define arrays.

The 'in' Keyword

The 'in' keyword determines if a given value matches any value in the list or not. The syntax of the keyword is

```
test_expression [not] in (expression [,expression...])
```

Code 2-22 "in" Keyword

For example, the example below, checks if any of the expressions 'OBSERVATION' and 'SIGNAL' is equal to test_expression which is 'command'.

```
select * from RFIDEvent where command in ('OBSERVATION'
                        ,'SIGNAL')
```

Above statement is equivalent to,

```
select * from RFIDEvent where command='OBSERVATION' or com-
                        mand='SIGNAL'
```

The 'between' Keyword

The 'between' keyword is used to specify the range of the test. Its syntax is,

```
test_expression [not] between begin_expression and
                        end_expression
```

Code 2-23 "between" Keyword

The 'like' Keyword

The like keyword provides facility of standard SQL pattern matching by 'like' keyword. SQL Pattern matching matches any single character by '_' and arbitrary characters of numbers by '%'.

Its syntax can be understood from following example.

```
select * from PersonalLocationEvent where name like
                        '%Jack%'
```

Code 2-24 "like" Keyword

The 'regexp' Keyword

The regexp keyword is form of matching based on regular expressions which yield a String type or numerical result. Its syntax is,

```
test_expression [not] regexp pattern_expression
```

Code 2-25 "regexp" Keyword

2.2.3. EPL Functions

EPL functions are classified as Single row functions and Aggregate Functions which are explained below (ESPER 2011).

Single Row Functions

Single row functions returns a single value for every single result row. Single row functions used in EPL are outlined in following table.

Table 2-5 Single Row Functions

Single Row Function	Result
Case value { When compare_value then result [when compare_value then result] [else result] end	Returns result where the first value equals compare_value
Case When condition then result [when condition then result] [else result] End	Returns result for the first condition that is true.
Max (expression, expression [, expression ...])	Returns the highest numeric value.
Min (expression, expression [, expression ...])	Returns the lowest numeric value.

Aggregate Functions

The EPL aggregated functions are summarized in table below.

Table 2-6 Aggregated Function

Aggregate Functions	Description
Sum([all | distinct] expression)	It sums all the (distinct) values in the expression.
Avg([all | distinct] expression)	It gives average of all the (distinct) values in the expression
Count([all |distinct] expression)	It counts the total non-null (distinct) values in expression.
Count (*)	Number of events
Max([all | distinct] expression)	Highest (distinct) value in the expression
Min([all | distinct] expression)	Lowest (distinct) value in the expression

2.3. Event Pattern Language

Event pattern are used to match event or multiple events whenever event matches definition of pattern (ESPER 2011).

There are 4 types of pattern operators,

1. Operator responsible for repeating pattern sub-expression: every, every-distinct, [num] and until.
2. Logical Operators: and, or
3. Temporal operators that functions on a specific event order: → (followed by)
4. Guards control the life cycle of sub-expression: timer:within, timer:withinmax and while.

This section discusses syntax, operators, pattern guards and pattern observers of Event Pattern Language in detail with examples (ESPER 2011).

Pattern Syntax

Following pattern matches on every Car event in which value of petrol event property is less than 1 litre.

```
every (ErrorMessage=Car(petrol<1)
```

Code 2-26 Pattern Syntax

In the above example, *every* specifies that pattern should match for every event, not just the first one. Within brackets, there is a filter expression matches only events having low petrol level.

Patterns in EPL

A pattern can appear anywhere in from clause in EPL statement. Pattern also can be used in combination with where clause, group by clause and having clause.

The following statement selects total price per customer over events (ServiceOrder event is followed by ProductOrder event for same customer id within 1 minute) occurring in last 2 hours and where clause is used to filter on name and sum of the price is greater than 100.

```
select a.custId, sum(a.price + b.price)
 from pattern [every a=ServiceOrder ->
b=ProductOrder(custId = a.custId) where timer:within(1
            min)].win:time(2 hour)
  where a.name in ('Repair', b.name)
            group by a.custId
  having sum(a.price + b.price) > 100
```

Code 2-27 Pattern Example

Pattern Operators

Every

The every clause is used to restart sub-expression after it evaluates to true or false. Without every operator, the sub-expression stops after it qualifies to true or false for the first time.

Consider following statement.

```
every A -> (B -> C) where timer:within(1 hour)
```

The sequence of events arriving is,

$$A_1\ A_2\ B_1\ C_1\ B_2\ C_2$$

This pattern always keeps on looking for A events. After A_1 arrives, the pattern keeps A_1 in memory and starts looking for B events. Also at the same time, it keeps on looking for more A events. After A_2 arrives, the pattern keep A_2 in memory and start keep searching for any B event or any A event.

After arrival of A_2 event, there are 3 sub-expressions active.

- The first sub-expression having A_1 in memory looking for B events.
- The second sub-expression having A_2 in memory looking for B events.
- The third sub-expression looking for more A events.

The pattern matches upon arrival of C_1 event for the combination of (A_1, B_1, C_1) and (A_2, B_1, C_1), with a condition that $B_1$and C_1 arrives within an hour of A_1 andA_2.

Rest of pattern operators are explained in table below.

Table 2-7 Pattern Operators

Pattern Operator	Syntax	Description
Every-Distinct	every-distinct(*distinct_value_expr* [, *distinct_value_exp*[...][, *expiry_time_period*])	Similar to every operator, every-distinct restarts sub-expressions after it evaluates to true or false. The only difference is that every-distinct eliminates duplicate results.
Repeat	[*match_count*] *repeating_subexpr*	The repeat operator continues to fire when pattern sub-expression evaluates to true for a given number of times.
Repeat-Until	[*range*] *repeated_pattern_expr* until *end_pattern_expr*	The repeat-until operator takes in consideration extra control over repeated matching by including a second sub-expression that ends the repetition.
And	A and B	In case of *and* operator, both nested pattern expressions need to be true.
Or	A or B	Pattern matches when either one of the expression turns true.
Not	(A → B) and not C	This operator negates the truth value of an expression.
Followed By	A → B	This operator specifies that first left hand value must true and only then right hand is evaluated.

Pattern Guards

Pattern Guards are where condition which specify the lifetime of a subexpression. Different pattern guards are explained below in Table 2.8.

Table 2-8 Pattern Guards

Pattern Operator	Syntax	Description
timer:within	timer:within(*time_period_expression*)	The *timer:within* serves as the stopwatch. If any certain pattern expression does not get true in the time specified in the timer:within, it is stopped and permanently false.
timer:withinmax	[*match_count*] *repeating_subexpr*	The timer:withinmax is similar to timer:within and has a additional counter to count number of matches. The sub-expression ends when stopwatch ends or counter reaches maximum value.
while	while (*guard_expression*)	The pattern sub-expression keeps on matching events until the value of guard_expression remains true. As value of guard_expression turns false, the pattern sub-expression ends.

Pattern Observers

The main function of pattern observers is to observe timer events. Different types of pattern observers are explained below in Table 2.9.

Table 2-9 Pattern Observers

Pattern Operator	Syntax	Description
timer:interval	A → timer:interval (10 seconds)	The timer:interval observes the time based events for the specified time and then truth value of observer is set to true. In the statement below, after arriving of A, observers waits for 10 seconds and then indicate that pattern is matched.
timer:withinmax	timer:at (minutes, hours, days of month, months, days of week, seconds])	The timer:at sets any expression true after specified time.

2.3.1. Example 1

Let us suppose that a production line in an industry manufactures 3 types of workpieces of colour red, black and silver. Following pattern irrespective of sequences of production of workpieces generates events whenever it matches any event.

```
on pattern [every red -> (black -> silver) while
                (red_counter1 < 11)]
       set red_counter1 = red_counter1 + 1
                          or
on pattern [every red -> (silver -> black) while
                (red_counter2 < 11)]
       set red_counter2 = red_counter2 + 1
                          or
on pattern [every black -> (red -> silver) while
              (black_counter1 < 11)]
     set black_counter1 = black_counter1 + 1
                          or
on pattern [every black -> (silver -> red) while
              (black_counter2 < 11)]
     set black_counter2 = black_counter2 + 1
                          or
on pattern [every silver -> (red -> black) while (sil-
                ver_counter1 < 11)]
    set silver_counter1 = silver_counter1 + 1
                          or
on pattern [every silver -> (black -> red) while (sil-
                ver_counter2 < 11)]
    set silver_counter2 = silver_counter2 + 1
```

For example the sequence of the workpieces production is

red1 black1 silver1 black2 red2 red3 black3 silver2 silver3

When red1 is received, then sub-expressions *(every a=red -> (black(id= red.id -> silver(id= red.id)) while (red.id < 11))* and *(every a=red -> (silver(id= red.id -> black(id= red.id)) while (red.id < 11))* matches it and waits for either black1 or silver1. But black1 comes earlier than silver1, so sub-expression *(every a=red -> (black(id= red.id -> silver(id= red.id))* matches it and then waits for silver1. Whenever silver1 arrives, a complex event is sent. This event matching continues until 10 pieces have been produced.

2.3.2. Example 2

Let us take another example of a production line in an industry which produces 3 types of workpieces of colour red, black and silver. Following pattern generates an event whenever it matches any pattern. This pattern keeps on matching events until 10 workpieces of all colours are produced within 300 seconds regardless of sequence of production of workpieces.

```
(every red -> (black -> silver)) where timer:withinmax(300
                        seconds,10)
                             or
(every red -> (silver -> black)) where timer:withinmax(300
                        seconds,10)
                             or
(every black -> (red -> silver)) where timer:withinmax(300
                        seconds,10)
                             or
(every black -> (silver -> red)) where timer:withinmax(300
                        seconds,10)
                             or
(every silver -> (red -> black)) where timer:withinmax(300
                       seconds,10) or
(every silver -> (black -> red)) where timer:withinmax(300 sec-
                          onds,10)
```

For example the sequence of the workpieces generation is

red1 black1 silver1 black2 red2 red3 black3 silver2 silver3

When red1 is received, then sub-expressions *(every a=red -> (black(id= red.id -> silver(id= red.id)) while (red.id < 11))* and *(every a=red -> (silver(id= red.id -> black(id= red.id)) while (red.id < 11))* matches it and waits for either black1 or silver1. But black1 comes earlier than silver1, so sub-expression *(every a=red -> (black(id= red.id -> silver(id= red.id))* matches it and then waits for silver1 . Whenever silver1 arrives, a complex event is sent. This event matching continues until 10 pieces have been produced.

2.4. Formal Methods Theory

2.4.1. Petri Nets

Petri Nets (PN) consists of places, transitions and flowarcs (Murata 1989). The flowarcs are used to connect places and transitions. A place is depicted by a circle and is used to show distributed state of the system. A transition is depicted by a bar or a box and is used to denote activity of the system. A place is an input place to a transition if a flowarc exists from place to transition. On the other hand, a place is an output place if there is a flowarc from transition to place.

Each place may hold positive number of tokens which are depicted by black dots. In order to understand PN, consider the following figure of a Machine-Conveyor system.

In the example below, a part is transferred from a machine to a conveyor of 1 location.

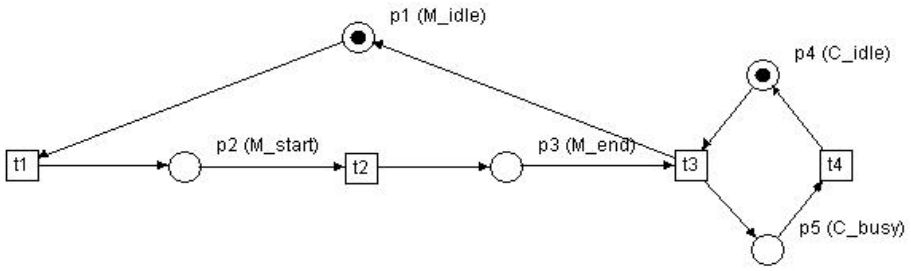


Figure 2-10 Machine-Conveyor System

The elements of this system are places ({p1, p2, p3, p4, p5}), transitions ({t1, t2, t3, t4}) and flow arcs ({(p1, t1), (p2, t2), (p3, t3), (p4, t3), (p5,t4), (t1, p2), (t2, p3), (t3, p1), (t3, p5), (t4, p4)}). The machine is in idle state if there is a token in place p1 (M_idle) and conveyor is in idle state if there is a token in place p4 (C_idle). Places p2 (M_start) denotes that machine has started processing part and place p3 (M_end) signifies that machine has ended processing part. If there is a token in place p5 (C_busy), then it shows that part has been placed from machine to conveyor. At any given time instance, current tokens distribution is called Petri Net marking and it represents the current state of the system.

Richard Zurawski and MengChu Zhou (1994) has defined Petri Nets formally as,

$$\mathbf{PN} = (\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O}, \mathbf{M_0}) \quad \mathbf{(1)}$$

where

P is a finite set of places,

T is a finite set of transitions,

I is a finite set of directed arcs from places to transitions,

O is a finite set directed arcs from transitions to places, and

M_0 is the initial marking

A transition is enabled if its input place contains at least that number of tokens equal to weight of flowarc from place to transition. For example, in Figure 2.5, transition t1 is enabled as place p1 has a token. The transitions also govern the flow of tokens by a rule called firing. Firing rule states that if transition is enabled and firing of tokens happens, then number of tokens equal to weight of transition connecting p to t is removed from input place. Then, number of tokens equal to weight of transition from t to p is deposited in output place.

The flow of tokens can be also explained graphically by Incidence matrix. In the Incidence matrix, columns represent transitions and rows represent places. The Incidence matrix of Figure 2.5 is shown in Table 2.1. Negative elements in matrix signify place-transition flowarc e.g. W[p1][t1]= -1 and it means that there is a flowarc from p1 to t1. Positive elements in matrix represent flowarcs from transition to place e.g. W[t1][p2] and it means that there is a flowarc from t1 to p2.

Table 2-10 Incidence Matrix for Figure 2.5

t1	t2	t3	t4	W
-1	0	+1	0	**p1**
+1	-1	0	0	**p2**
0	+1	-1	0	**p3**
0	0	-1	+1	**p4**
0	0	+1	-1	**p5**

After firing of pre-defined sequence of transitions $S_{transitions}$ from any intial marking M_0, marking M at that state can be obtained by following State Equation.

$$\mathbf{M = M_0 + W.S} \quad \textbf{(2)}$$

where S is the characteristic vector corresponding to $S_{transitions}$. For PN in Figure 2.5, marking after firing transition t1 can be calculated as,

$$\begin{bmatrix}0\\1\\0\\1\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\\0\end{bmatrix} + \begin{bmatrix}-1 & 0 & +1 & 0\\ +1 & -1 & 0 & 0\\ 0 & +1 & -1 & 0\\ 0 & 0 & -1 & +1\\ 0 & 0 & +1 & -1\end{bmatrix} \cdot \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

Similarly, reachability graph or state space of Figure 2.5 can also be derived algebraically as shown below.

$$M_0 = (1\,0\,0\,1\,0) \xrightarrow{t1} M_1 = (0\,1\,0\,1\,0) \xrightarrow{t2} M_2 = (0\,0\,1\,1\,0) \xrightarrow{t3} M_3(1\,0\,0\,0\,1)$$

$$\uparrow \quad \_\_\_\_\_ t4 \_\_\_\_\_ \quad \rfloor$$

With the help of Petri Nets, many qualitative characteristics of system can be checked: reachability, boundedness and safeness, conservativeness, liveness etc. (Murata 1989).

Hence, Petri Nets is a graphical tool which has very powerful communication medium between users, engineers and customers (Richard Zurawski and MengChu Zhou 1994). As it is a mathematical tool, formal analysis of any model is possible. Also many other properties like deadlock avoidance, concurrent operations, suitable synchronization among resources and mutual exclusion of shared properties can also be studied. Joanne Bechta Dugan and Kishore S. Trivedi (1989), Nancy G. Leveson and Janice L. Stolzy (1987), Fevzi Belli and Karl-E Grosspietsch (1991) have demonstrated how to model real-time fault tolerant systems in Petri Nets. A. Chaillet, M. Combacau and M. Courvoiser (1993), V.S. Srinivasan and M.A. Jafari (1991) and Valette R., Cardoso J., Dunois D. have studied fault detection and in-process monitoring using Petri Nets.

2.4.2. Timed Net Condition Event System

Timed Net Condition/Event Systems (TNCES) (Rausch M., Hanisch H.-M. 1995) exist in several forms and have been utilized in modelling, verification and validation of control systems. Modules used in modelling phase are pre-tailored and are used over and over again.

In TNCES, contrary to classical Petri Nets (PN) approach, firing of a token does not depend on current marking but also on incoming condition and event signals. Transitions are forced to be fired only if they are allowed by marking and condition signals.

Inputs and outputs are of two types,

1. Condition inputs/outputs which are responsible for carrying state information.
2. Event inputs/outputs which are responsible for carrying transition information.

TNCES can also be expressed in a tuple as follows by (Rausch M., Hanisch H.-M. (1995).

$$\mathbf{TNCES} = \{\mathbf{P}, \mathbf{T}, \mathbf{F^+}, \mathbf{F^-}, \mathbf{M_0}, \boldsymbol{\psi}, \mathbf{CN}, \mathbf{EN}, \mathbf{DC}\} \quad \textbf{(3)}$$

where:

- $\mathbf{P} \coloneqq \{p_1, p_2, \dots, p_n\}$ a set of places
- $\mathbf{T} \coloneqq \{t_1, t_2, \dots, t_m\}$ a set of transitions
- $\mathbf{F} \subseteq (\mathbf{P\ X\ T}) \cup (\mathbf{T\ X\ P})$ a finite set of flow arcs between places and transitions
- $\mathbf{CN} \subseteq (\mathbf{P\ X\ T})$ a finite set of condition arcs
- $\mathbf{EN} \subseteq (\mathbf{T\ X\ T})$ a finite set of event arcs
- M_0 an initial marking
- ψ the input/output structure of a TNCES module

$$\boldsymbol{\psi} = \{\mathbf{C^{in}}, \mathbf{E^{in}}, \mathbf{C^{out}}\ \mathbf{E^{out}}, \mathbf{B_c}, \mathbf{B_e}, \mathbf{C_s}, \mathbf{D_t}\}$$

where:

  - $\mathbf{C^{in}}$ a finite set of TNCES module condition input signals
  - $\mathbf{E^{in}}$ a finite set of TNCES module event input signals
  - $\mathbf{C^{out}}$ a finite set of TNCES module condition input signals
  - $\mathbf{E^{out}}$ a finite set of TNCES module event input signals
  - $\mathbf{B_c} \subseteq \mathbf{C^{in}\ X\ T}$ a set of TNCES module input condition arcs
  - $\mathbf{B_e} \subseteq \mathbf{E^{in}\ X\ T}$ a set of TNCES module input event arcs
  - $\mathbf{C_s} \subseteq \mathbf{P\ X\ C^{out}}$ a set of TNCES module output condition arcs
  - $\mathbf{D_t} \subseteq \mathbf{T\ X\ E^{out}}$ a set of TNCES module output condition arcs

There are some time constraints related to flow arcs prior to transitions ($F^- \subseteq$ (P X T)):

$$\mathbf{DC} = \{\mathbf{DR}, \mathbf{DL}, \mathbf{D_0}\}$$

where:

- **DR:** the minimum time that a token should remain at a place before enabled transition(s) can fire.

- **DL:** the maximum time that a place should keep a token (if all other necessary conditions for firing a transition are fulfilled).
- **D_0:** the initial set of clocks associated with the places (to record local current time associated with places).

An example of a TNCES system is described in Figure 2.11. A module with name "Machine-Conveyor System" has places ({p1, p2, p3, p4, p5}), transitions ({t1, t2, t3, t4}) and flow arcs ({(p1, t1), (p2, t2), (p3, t3), (p4, t3), (p5,t4), (t1, p2), (t2, p3), (t3, p1), (t3, p5), (t4, p4)}). There is also a condition input ({c1}), an event input ({e1}), a condition output ({co1}) and an event output ({eo1}). The condition arc ({(ci1, t1), (p4, eo1)}) connects condition input and transition/places and condition output. The event arc ({(ei1, t3), (t4, eo1)}) connects event input and transition/ transition and event output.

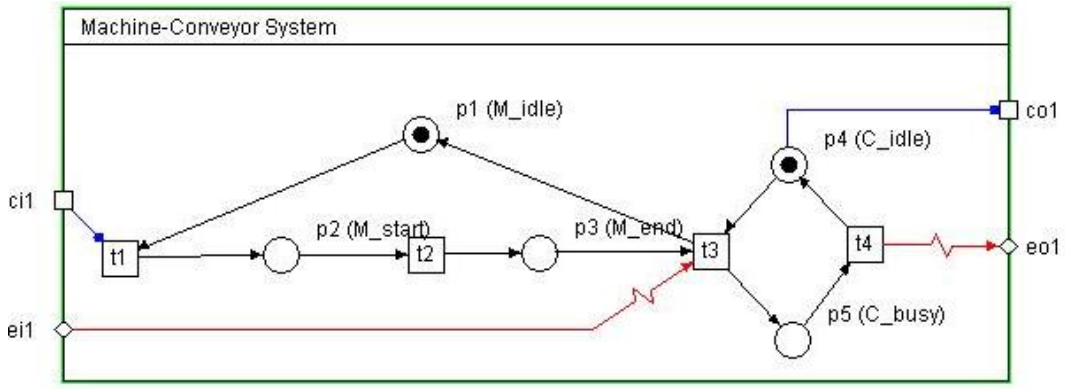


Figure 2-11 Machine Conveyor System

From Figure 2.6, it can be seen that elements of tuple given in (3) are listed below.

- **P** = ({p1, p2, p3, p4, p5})
- **T**= ({t1, t2, t3, t4})
- **F**= ({(p1, t1), (p2, t2), (p3, t3), (p4, t3), (p5,t4), (t1, p2), (t2, p3), (t3, p1), (t3, p5), (t4, p4)})
- $\mathbf{C^{in}}$ = {ci1}
- $\mathbf{E^{in}}$ ={ei1}
- $\mathbf{C^{out}}$ ={co1}
- $\mathbf{E^{out}}$ ={eo1}
- $\mathbf{B_c}$={(ci1, t1)}; B_c={(ei1, t2)}; B_c={(p3, co1)}; B_c={(t4, eo1)};
- $\mathbf{D_0}(\boldsymbol{P})$ = {0, 0, 0, 0, 0}; DR and DL matrices of size N_{places}X $N_{transitions}$.

Normally, there are two classes of transitions, i.e. spontaneous transitions and forced transitions. Spontaneous transitions are those transitions which do not have any incoming event signal and forced transition are those transitions which have at least one incoming event signal. But in TNCES, there is third type of transition which is forced to be fired by time.

Lobov A., Popescu C. and Lastra J. L. M. (2006) described the possible approaches in applying and modelling of any system using formal methods. Two main approaches are model-based code generation and code- based model modelling. First approach is common and widely used. In this approach, in the first step, formalization of the problem is done, then proof of correctness is achieved and in the last, implementation is done based on the model.

Hanisch, H-M., Thieme, J., Luder, A., Wienhold, A (1997) had presented a PLC behaviour modelled in TNCES in which all transitions are dynamically fired as soon as they are enabled. Whole model works in a way that binary inputs from sensors are read and copied into process input storage. Then control program statements are executed and output of the output storage are pushed out by firing relevant transition. Models of AND, OR and Timers programmed by TNCES are also presented.

For automatic translation of PLC code into TNCES, a translator in MATLAB is developed. It is based on determining of elements in Instruction List (IL) and this is done by recognisers. Depending on the input characters, recogniser will change its state. For example, in first statement detected, it will be read completely and it will be determined whether it is an operator and operand and depending on that, recogniser will change its state.

Puttonen J., Lobov A. and Martinez Lastra J. L. (2008) had extended the idea of TNCES to modelling of hybrid systems. The resulting system after combining TNCES with hybrid Petri nets formalism is called hybrid TNCES or HNTCES. In this type of formalism, discrete and continuous elements affect each other directly due to which discrete change in the continuous variables is possible. Secondly, ordinary event signals may not be used with continuous transitions as continuous transitions cannot fire at discrete time intervals. Last, time values related to the flow arcs cannot be applied to continuous transitions rather than only to discrete transitions.

In another journal by Popescu C., Lobov A., Martinez Lastra J.L., Cavia Soto M. (2008), TNCES representation of Split, Split+Join, Choice, If-Then-Else, Repeat-While, Repeat-Until, AND logical connector and OR logical connector is explained in detail.

2.5. Utilizing Formal Methods in Factory Automation

In factory automation, formal methods are used to verify and validate any system and ensure smooth manufacturing process such as scheduling and planning production and prevention of deadlock. This section discusses how formal methods verify and validate systems in factory automation and coordinate smooth production in automation systems.

2.5.1. Formal Verification and Validations of Systems

Formal verification is a process to check whether system satisfies a set of properties that are derived from its specification and standard requirements (Are we building the thing right?). Formal validation makes sure that whether the formal model is consistent with

the informal conception of design and product does that user actually requires. (Are we verifying the right thing?).

Mainly there are two formal verification techniques: model checking (Clarke et. al 2001) and theorem proving (Duffy D.A. 1991). In model checking, specification of system are checked automatically on a finite system model based on Petri Nets, automata etc.

In model checking, the properties of a model of any system can be derived from the requirements of the system. Those properties are studied by applying Computational Tree Logic CTL (Clarke, 2001). CTL has two path quantifiers, "A" or "□" which represents for all paths" and "E" or "◊" which signifies "for some path". With the aid of these quantifiers, properties can be defined for all paths or some specific path. Moreover, there are five other temporal operators.

- **G** (Globally) operator represents that the property exists for all possible on the path.
- **X** (Next) operator shows that the property exists for next state on the path.
- **F** (Finally) operator specifies that property eventually holds somewhere on the subsequent path.
- **U** (Until): ψ U Φ – expects that ψ has holds at least until Φ, which holds at the current or a future position.
- **R** (Release) ψ R Φ – indicates that Φ holds until and including the point where ψ first becomes true. If ψ never becomes true, Φ must remain true forever.

Following figure also explains procedure of model checking.

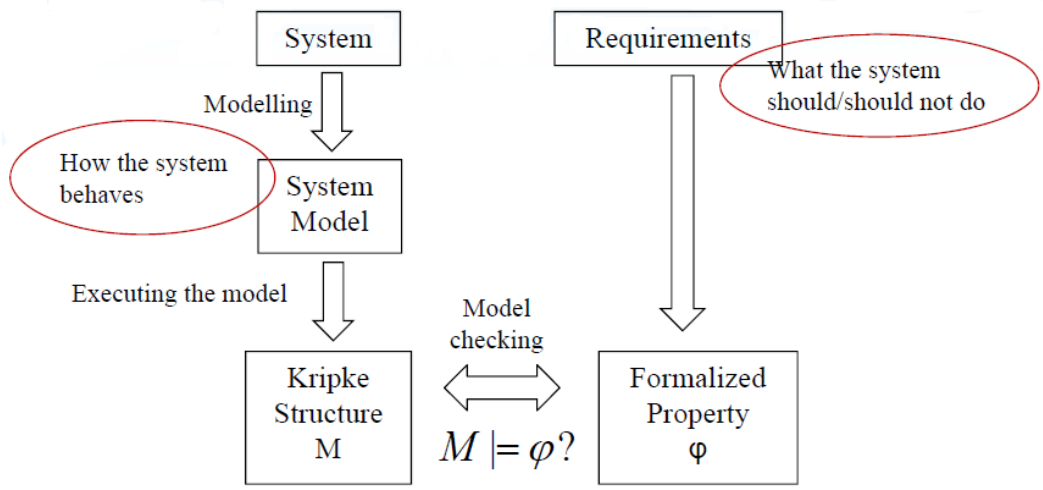


Figure 2-12 Model Checking (Clarke 2001)

Kripke structure for any set of atomic propositions can be defined as:

$$M = \langle S, S_o, R, L \rangle$$

where:

- S– a finite set of states
- S_0 – a set of initial states, $S_0 \subseteq S$
- $R: S \times S$ – is a transition relation between states
- $L: S \rightarrow S^{AP}$- is a function which assigns each state with the atomic proposition which hold in that state.

The CTL formulae can consist of different combination temporal operators and path quantifiers. In addition, Boolean operators are also allowed.

The other technique "Theorem proving" considers a mathematical model to formalize system and its expected properties. Inference rules are then used to verify the properties of the system through its axioms.

2.5.2. Planning and Scheduling

Planning is deciding what actions should be taken to achieve some set of objectives. On the other hand, scheduling is deciding how to perform a given set of actions using a limited number of resources in a limited amount of time in order to enhance efficiency, production, quality and minimization of costs.

There are three techniques of scheduling which are found in literature which are completely reactive scheduling, predictive-reactive scheduling and robust pro-active scheduling.

Completely reactive scheduling techniques execute real time schedules through heuristic dispatching rules and bidding mechanisms rather than creating schedules beforehand (Sousa P. and Ramos C. 1999).

Predictive/reactive scheduling is a process of updating previously created schedules in an iterative manner (Jain A.K. and Elmaraghy H. A. 1997).

Robust proactive scheduling is a process of creation of robust and efficient schedules which satisfy performance requirements in a dynamic environment.

2.5.3. Deadlocking Prevention

Deadlocks are situations in which system or a part of it remains indefinitely blocked and cannot terminate its task (Fanti M. P., Zhou M.C. 2004). The main cause of deadlock is inappropriate allocation of resources to concurrent executing processes.

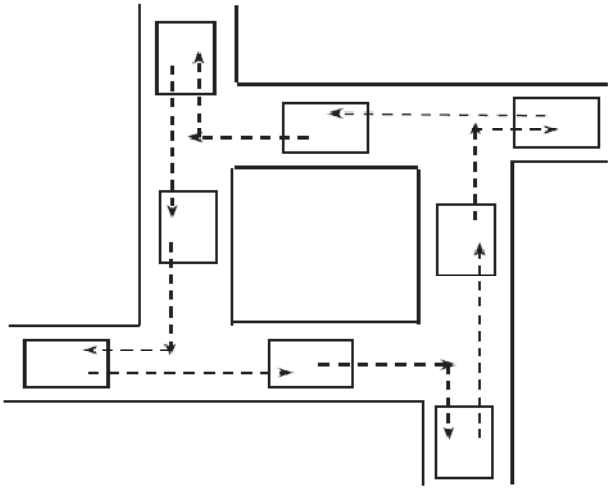


Figure 2-13 Deadlock Example (Fanti M. P., Zhou M.C. 2004)

There are four conditions for a deadlock to occur. First condition is called mutual exclusion in which tasks claim exclusive control of the resource. Second condition is called No pre-emption in which resources cannot be forcibly removed from tasks holding them until resources are used for completion. Third condition is known as Wait for condition in which processes hold resources allocated to them while waiting for additional ones. The last condition is called Circular wait in which a circular claim of tasks exists, in such a way that each task holds one or more resource that are being requested by the next task in the claim.

In an Automated Manufacturing System, first three conditions always hold. But deadlock is prevented if last condition fails to hold. There are three strategies to prevent deadlocks: Deadlock prevention methods, Deadlock detection and recovery methods and Deadlock avoidance methods.

In Deadlock prevention methods, circular waits among concurring jobs at the design stage are prevented. In Deadlock detection and recovery methods, there is a monitoring mechanism to detect deadlock. If deadlock is detected, there is a recovery procedure to get system out of deadlock. In Deadlock avoidance methods, proper operational control of part flow is ensured to prevent condition of circular wait. To avoid circular waits, the interaction of resources and tasks are normally depicted through graphs (Wysk R.A, Yang N.S, Joshi S. 1991) (Kim C.O., Kim S.S. 1997) or Petri Nets (Banaszak Z.A, Krogh B.H., 1990) (Viswanadham N., Narahari Y., Johnson T.L. 1990) (Hyun Joong Yoon, Doo Yong Lee 2000)

3. METHODOLOGY

For the purpose of modelling CEP into TNCES, we need to have TNCES model of Factory Floor and EPL statements. Then, events generated by Factory Floor are processed by TNCES module of EPL statements to produce complex events. TNCES module of EPL statements contains model of our desired EPL statement and that statement utilizing the concept of CEP, extracts meaningful events out of simple events from Factory Floor.

Modelling of Factory Floor is out of scope of this thesis work and very vast research on this topic can be found in literature (Descrochers A. 1989) (Lobov A. 2008) (Popescu C. 2009). On the other hand, modelling of EPL statements is done utilizing algorithm presented in next section. For modelling and validation purpose, NCES Editor and NCES Analyzer environment of Movida tools is used as a toolset (MOVIDA).

Section 3.1 explains composition rules needed to represent EPL statements into TNCES and Section 3.2 describes the algorithm devised to represent EPL statement into TNCES following rules given in Section 3.1. Section 3.3 and Section 3.4 presents TNCES models of some clauses of EPL and event pattern language respectively.

3.1. Composition Rules of representing an EPL statement into TNCES

The first important step in transferring EPL example into TNCES is dividing whole EPL statement into prominent and generalized TNCES modules. Second step is to compose sequence of events between modules. After identifying event sequence, next step is to define task of each module and their internal working is represented in TNCES. Last step is to connect modules via event and condition arcs in their correct order of sequence.

Let us consider an example below.

```
select * from MyEventStream
```

In this example, possible generalized TNCES modules can be Select, From, Properties and EventStreams. Select module initiates the Select construct and From module allows selection of event streams or named windows from which we want to select desired properties. In this example, we want to observe properties from events related to stream ‘MyEventStream’.

Properties module selects desired properties. For example, in this example, we are selecting all properties in the event stream by using asterisk *.

The desired event stream or window, from which we want to select properties of events, can be selected in module “EventStreams”. In the example, we have only one stream called ‘MyEventStream’.

The algorithm presented in next section takes in account these composition rules and is used to represent any EPL and event pattern language statement.

3.2. Algorithm for representing an EPL statement into TNCES

EPL statements are represented into TNCES using Algorithm 3.1 and results are given in next sections. Algorithms for translating *Select* clause, *From* clause, *Every* clause, *Followed By* clause, *timer_within* clause and *Repeat* clause are given separately in Algorithm 3.2, Algorithm 3.3, Algorithm 3.4, Algorithm 3.5, Algorithm 3.6 and Algorithm 3.7 respectively and are referred in Algorithm 3.1.

Algorithm 3-1 Generating TNCES model for an EPL statement

Generating TNCES model for an EPL statement	
Required Data:	
vecSelect	A vector of TNCES modules representing Select module. Initial value is empty.
vecProperties	A vector of TNCES modules representing Properties module. Initial value is empty.
vecFrom	A vector of TNCES modules representing From module. Initial value is empty.
vecEventStreamEngine	A vector of TNCES modules representing EventStreams module. Initial value is empty.
vecEventMatching	A vector of TNCES modules representing EventMatching module. Initial value is empty.
vecExpressions	A vector of TNCES modules representing Expressions module. Initial value is empty.
vecPattern	A vector of TNCES modules representing Pattern module. Initial value is empty.
Inputs:	
EPL statement	EPL statement containing event processing language and event pattern language statements.
Outputs:	
TNCES Model	TNCES module representing EPL statement
Algorithm Steps:	
1)	**while** EPL statement has clauses **do**
2)	Generate TNCES module representing Processor (tncesProcessor);
3)	**for** a Select clause **do**
4)	Apply algorithm for translating Select clause into TNCES (Algorithm 3.2);
5)	**od;**
6)	Add SelectTNCES to vecSelect;
7)	**for** all Properties in Select Clause **do**
8)	Generate TNCES module representing Properties (PropertiesTNCES);

9)	Add PropertiesTNCES to tncesProcessor;
10)	Define event inputs SelectProperty;
11)	Define event inputs for each property in the Select clause;
12)	Interconnect event inputs and PropertiesTNCES module;
13)	**if** Properties has Patterns, Operators, Functions
14)	Generate TNCES module for each Property representing Patterns/Operators/Functions to select (EventMatchingTNCES) where EventMaching is replaced by specific name of Pattern/Operators/Functions;
15)	Add EventMatchingTNCES to tncesProcessor;
16)	Define event inputs and condition outputs and related event conditions arcs to modify and observe values of EventMatchingTNCES;
17)	**fi;**
18)	Add EventMatchingTNCES to vecEventMatching;
19)	**od;**
20)	Add PropertiesTNCES to vecProperties;
21)	**for** a From Clause **do**
22)	Apply algorithm for translating From clause into TNCES (Algorithm 3.3);
23)	**od;**
24)	Add FromTNCES to vecFrom;
25)	**for** each Stream_Def Clause **do**
26)	Generate TNCES module to capture simple events (EventStreamEngineTNCES) where EventStream is replaced by name of Event Stream;
27)	Add EventStreamsTNCES to tncesProcessor;
28)	Define event inputs to modify values of EventStreamEngineTNCES;
29)	Interconnect event inputs and EventStreamEngineTNCES module;
30)	**od;**
31)	Add EventStreamEngineTNCES to vecEventStreamEngine;
32)	**for** each of where, group by, having, output, order by Expression **do**
33)	Generate TNCES module representing any of the Expression above (ExpressionTNCES);
34)	Add ExpressionTNCES to tncesProcessor;
35)	Define event inputs and condition outputs and related event conditions arcs to modify and observe values of ExpressionTNCES;
36)	**od;**
37)	Add ExpressionTNCES to vecExpression;
38)	**while** Pattern statement has clauses **do**
39)	Generate Pattern module representing Processor (patternProcessor);
40)	**for** a Every clause **do**
41)	Apply algorithm for translating Every clause into TNCES (Algorithm 3.4);
42)	**od;**
43)	**for** a Followed By clause **do**
44)	Apply algorithm for translating FollowedBy clause into TNCES (Algorithm 3.5);
45)	**od;**
46)	**for** a timer_within clause **do**
47)	Apply algorithm for translating timer_within clause into TNCES (Algorithm 3.6);
48)	**od;**
49)	**for** a Repeat clause **do**
50)	Apply algorithm for translating Repeat clause into TNCES (Algorithm 3.7);

51)	**od;**
52)	**for** all other Pattern Operators **do**
53)	Generate TNCES module representing Pattern Operators (PatternTNCES);
54)	Add PatternTNCES to patternProcessor;
55)	Define event inputs in PatternTNCES that would allow to modify the values of PatternTNCES;
56)	Interconnect event inputs and PatternTNCES module;
57)	Define condition outputs in PatternTNCES that would allow to observe the values of PatternTNCES;
58)	Interconnect condition outputs and PatternTNCES module;
59)	**od;**
60)	Add PatternTNCES to vecPattern;
61)	**do;**
62)	**do;**
63)	Interconnect vecSelect, vecProperties, vecFrom, vecEventStreamEngine, vecEventMatching, vecExpressions and vecPattern;

The following algorithm generates TNCES model of Select clause shown in Figure 3.1 (a). This is the module which initiates the Start clause and then ends the clause upon its completion. Therefore, we have two event inputs “Select” and “EndSelect” which serve the purpose. After starting Select module, we have to inform next module to start working and event output “SelectProperty” is for this reason. Next event output is “EndSelectConstruct” to stop all modules from operating whenever we want to end execution of EPL statement.

Algorithm 3-2 Generating TNCES model of Select Clause

Generating TNCES model of Select Clause	
1)	Generate TNCES module representing Select (SelectTNCES);
2)	Add SelectTNCES to tncesProcessor;
3)	Define event inputs Select and EndSelect in SelectTNCES that would allow to modify the input values of SelectTNCES;
4)	Define event outputs SelectProperty and EndSelectConstruct in SelectTNCES that would allow to modify the output values of SelectTNCES;
5)	Create places p1 and p2;
6)	Create transitions t1 and t2;
7)	Create flow arcs (p1,t1), (t1,p2), (p2,t2) and (t2,p1);
8)	Create event arcs (Select,t1), (EndSelect,t2), (t1, SelectProperty) and (t2, EndSelectConstruct);

Similarly, From clause is represented in TNCES utilizing algorithm given below. From module is shown in Figure 3.1 (b). From module starts working after desired property is specified in the Properties module with the help of event input “PropertySelected”. From module starts working with event input “From”. To inform next module “EventStream” to start functioning, we have event output “ FromEventStream”.

Algorithm 3-3 Generating TNCES model of From Clause

Generating TNCES model of From Clause	
1)	Generate TNCES module representing From (FromTNCES);
2)	Add SelectTNCES to tncesProcessor;
3)	Define event inputs PropertySelected and From in FromTNCES that would allow to modify the input values of FromTNCES;
4)	Define event output FromEventStream in FromTNCES that would allow to modify the output values of FromTNCES;
5)	Create places p1 and p2;
6)	Create transitions t1 and t2;
7)	Create flow arcs (p1,t1), (t1,p2), (p2,t2) and (t2,p1);
8)	Create event arcs (PropertySelected,t1), (From,t2) and(t2, FromEventStream);

Every module is a part of Event Pattern Language and it is developed using same rules as of EPL. The following algorithm generates TNCES module of Every clause which is shown in Figure 3.2. Whenever any sub-expression meets the criteria by Every keyword, pattern is restarted by Every keyword. To start Every module, we need one event input which receives events from Factory Floor and this input is called "A" in the following algorithm. Then we need one event input, so that Every module initiates working and we have event input "Start_A" for this purpose. Another event input is "Reset" which resets Every module on completion of EPL statement.

Algorithm 3-4 Generating TNCES model of Every Clause

Generating TNCES model of Every Clause	
1)	Generate TNCES module representing Every (EveryTNCES);
2)	Add EveryTNCES to patternProcessor;
3)	Define event inputs A, Reset1, Start_A and Reset2 in EveryTNCES that would allow to modify the input values of EveryTNCES;
4)	Define event outputs Event_A andEvent_A_match in EveryTNCES that would allow to modify the output values of EveryTNCES;
5)	Define condition output A_available in EveryTNCES that would allow to observe the output values of EveryTNCES;
6)	Create places p1, p2 and p3;
7)	Create transitions t1, t2, t3, t4 and t5;
8)	Create flow arcs (p1,t1), (t1,p2), (p2,t2), (p2, t5), (t2, p3), (p3, t3), (t3, p4), (p4, t4), (t4, p1) and (t5,p1);
9)	Create event arcs (Reset1,t1), (Start_A,t2), (Reset2, t3), (t1, Event_A) and (t2, Event_A_match);
10)	Create condition arcs (p2, A_available);

The algorithm 3.5 generates TNCES model of Followed By Clause and is shown in Figure 3.3. The main function of Followed By operator is to first evaluate left hand side expression to true and then right hand side. Therefore, we have two event inputs "Event_LHS" and "Event_RHS" to make sure that first LHS expression is executed to true and then RHS is evaluated. Another event input is "Reset" which resets whole

module whenever needed. Then, we have two event outputs “Start_LHS” and “Start_RHS” to indicate beginning of evaluation of LHS expression and RHS expression. Another event outputs “Complex Event” and “FollowedByFinished” signifies completion of pattern language and generation of complex event.

Algorithm 3-5 Generating TNCES model of Followed By Clause

Generating TNCES model of Followed By Clause	
1)	Generate TNCES module representing FollowedBy (FollowedByTNCES);
2)	Add FollowedByTNCES to patternProcessor;
3)	Define event inputs Reset, Event_LHS and Event_RHS in FollowedByTNCES that would allow to modify the input values of FollowedByTNCES;
4)	Define event outputs Start_LHS, Start_LHS, ComplexEvent and FollowedByFinishsed in FollowedByTNCES that would allow to modify the output values of FollowedByTNCES;
5)	Define condition outputs PatternSelected, LHS_available and RHS_available in FollowedByTNCES that would allow to observe the output values of FollowedByTNCES;
6)	Create places p1, p2, p3, p4, p5 and p6;
7)	Create transitions t1, t2, t3, t4, t5, t6, t7, t8, t9 and t10;
8)	Create flow arcs (p6,t6), (t6,p5), (p5,t1), (t1, p1), (p1, t2), (t2, p2), (p2, t3), (t3, p3), (p3, t4), (t4,p4), (p4, t5), (t5, p6), (p3, t10), (p2, t9), (p1, t8), (p5, t7), (t10, p6), (t9, p6), (t8, p6) and (t7, p6);
9)	Create event arcs (Reset,t7), (Reset, t8), (Reset, t9), (Reset, t10), (Event_LHS, t2) and (Event_RHS, t2);
10)	Create condition arcs (PatternSelected, t6), (LHS_available, t1) and (RHS_available, t3);

The algorithm given below translates timer_within clause in Event Pattern Language and timer_within module is displayed in Figure 3.4. In this module, we have one event input “Start_Timer_within” to specify starting of timer_within module. There are two event outputs “Timer_within_started” and “End_timer_within”, which signifies starting and ending of timer_within clause.

Algorithm 3-6 Generating TNCES model of timer_within Clause

Generating TNCES model of timer_within Clause	
1)	Generate TNCES module representing timer_within (timer_withinTNCES);
2)	Add timer_withinTNCES to patternProcessor;
3)	Define event input Start_timer_within in timer_withinTNCES that would allow to modify the input values of timer_withinTNCES;
4)	Define event outputs Timer_within_started and End_timer_within in timer_withinTNCES that would allow to modify the output values of timer_withinTNCES;
5)	Create places p1, p2 and p3;
6)	Create transitions t1, t2 and t3;
7)	Create flow arcs (p1,t1), (t1,p2), (p2,t2), (p2, t2), (t2, p3), (p3, t3) and (t3, p1);
8)	Create event arc (Start_timer_within,t1), (t1, Timer_within_started) and (t2, End_timer_within);

The following algorithm translates Repeat clause in Event Pattern Language. The resulted Repeat module is shown in Figure 3.5. There is an event input "Start_Repeat" and an event output "End_Check" to stipulates starting and finishing of Repeat operator. Another event input is "Event" which is used to receive events from Factory Floor.

Algorithm 3-7 Generating TNCES model of Repeat Clause

Generating TNCES model of Repeat Clause	
1)	Generate TNCES module representing Repeat (RepeatTNCES);
2)	Add RepeatTNCES to patternProcessor;
3)	Define event inputs Start_Repeat and Event in RepeatTNCES that would allow to modify the input values of RepeatTNCES;
4)	Define event output End_Check in RepeatTNCES that would allow to modify the output values of RepeatTNCES;
5)	Create places p1, p2 and p3;
6)	Create transitions t1, t2 and t3;
7)	Create flow arcs (p1,t1), (t1,p2), (p2,t2), (p2, t2), (t2, p3), (p3, t3) and (t3, p1);
8)	Create event arcs (Start_Repeat,t1) and (Event,t2);
9)	Create condition arcs (t3, End_Check);

3.3. Modelling EPL Constructs

The following constructs of EPL and event pattern language are represented into TNCES in this thesis. These are most commonly used constructs and by following algorithm presented in Table 3.1, any other construct can be translated into TNCES.

Table 3-1 Table of EPL Constructs translated in to TNCES

1.	Select
2.	From
3.	Every
4.	Followed By
5.	Repeat
6.	timer_within

The TNCES models of following logical constructs are taken from previous work. (Popescu C. 2009).

Table 3-2 TNCES modules taken from previous work

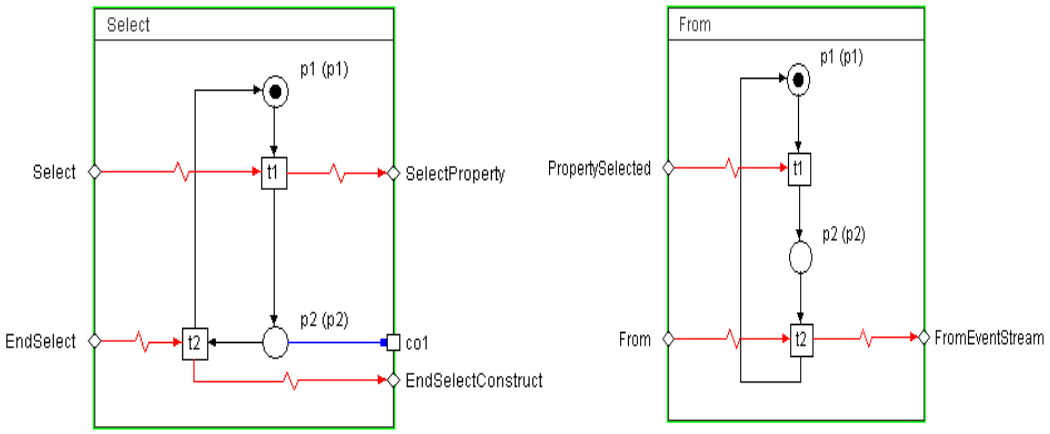
1.	AND
2.	OR
3.	NOT
4.	TimeCheck

Select Module

Select module initiates the Select construct and it is developed using Algorithm 3.2 explained in last section. It sends an event 'SelectProperty' to the next module 'Properties'. On the completion of the query, an event 'EndSelect' is received which ends the Select construct and an event 'EndSelectConstruct' is sent to module 'Properties'.

From Module

This module utilizes Algorithm 3.3 and allows selection of event streams or named windows from which we want to select desired properties. After selecting event stream, an event is sent to module associated with that event stream.



(a) (b)

Figure 3-1: (a) Select (b) From

Properties Module

In this module, desired properties of events can be selected. After receiving event 'SelectProperty', this module sends an event 'PropertySelected' to 'From' module. In case if Select Construct has been ended or truncated, an event 'EndSelectConstruct' is received, and in turns an event 'EndSelecting' is sent.

EventStreams Module

In this module, the desired event stream or window, from which we want to select properties of events, can be selected.

3.4. Modelling Event Pattern Language Constructs

Some of the most needed event pattern language constructs explained in Section 2.3 like every, followed by, timer:within, repeat are modelled in TNCES.

Every

The *every* operator restarts the pattern whenever the sub-expression qualified by *every* keyword evaluates to true or false. This module is created with the aid of algorithm given in last chapter in Algorithm 3.4.

The above module shown in Figure 3.2 keeps on receiving events by event input 'A' and keep them stored in place m (p2). The Every module is triggered by an event input

'Start_A' and then it sends an event 'Event_A_match' to the next module. There is one event input called 'Reset1' and it is used to reset token in m(p2). Another reset called "Reset2" and it is used to reset whenever it is turn for next module to match Events B.

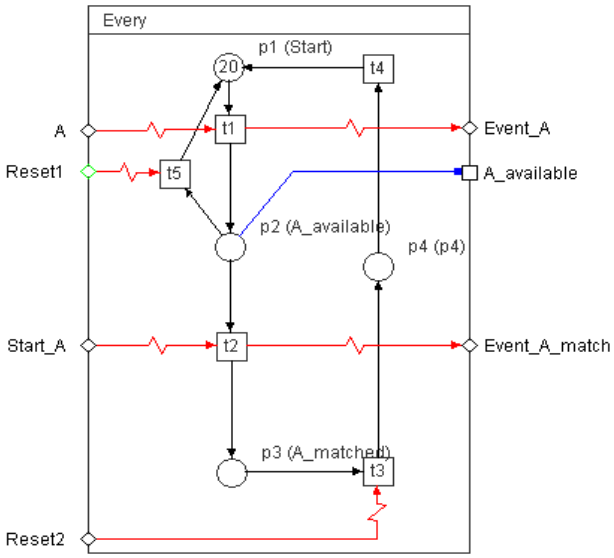


Figure 3-2 Every Module

Followed By

The followed by operator makes sure that first left hand expression must evaluate to true and then right hand expression is evaluated to match patterns.

The Followed By module is explained in Figure 3.3. The above module first waits for left hand expression to turn true by checking the condition 'LHS_available' and then it allows module on the left hand side to start matching events by sending an event 'Start_LHS'. When the next module evaluates to true by sending back an event 'Event_LHS', the *Followed By* module allows module on right hand side to match event. The *Followed By* module sends an event 'Start_RHS' and when right hand side matches event, it sends back an event 'Event_RHS'. Then the *Followed By* module comes to know that events are matches and it sends a complex event called 'ComplexEvent' and "FollowedByFinished" event in the end.

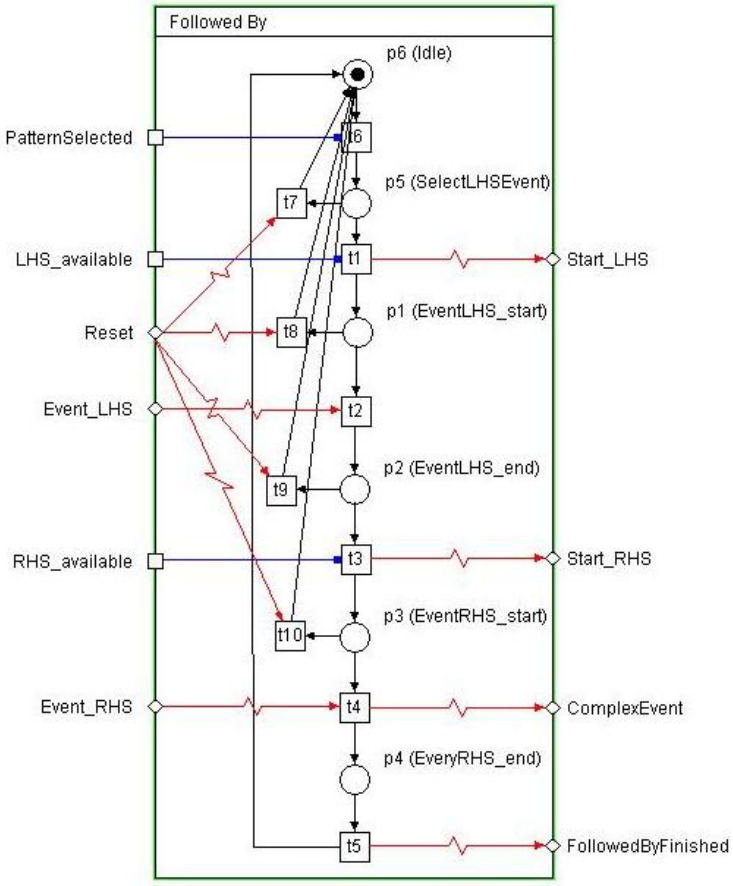


Figure 3-3 Followed By Module

Timer:within Pattern Guard

The timer:within acts like a stopwatch and it stops the pattern permanently after specified time.

The module shown below after getting being started by an event 'Start_timer_within' starts executing. And after 60 seconds, it sends an event called 'End_timer_within' and stops the pattern from further working.

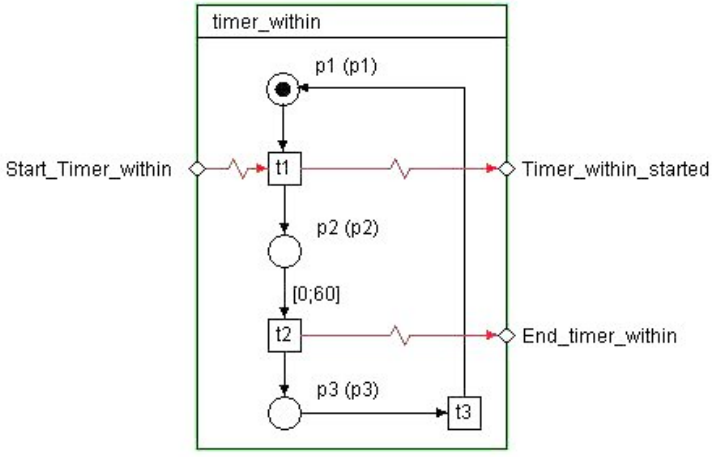


Figure 3-4 timer:within module

Repeat

The repeat operator fires whenever any pattern sub-expression evaluates to true for a given number of times.

The following ‘Repeat’ module waits for five sub-expressions to get true. The following module is triggered by an event ‘Start_Repeat’. Event input ‘Event’ receives events from the corresponding module which is generating complex events. After 5 sub-expressions get true, the following module stops executing and informs corresponding module by event ‘End_Check’.

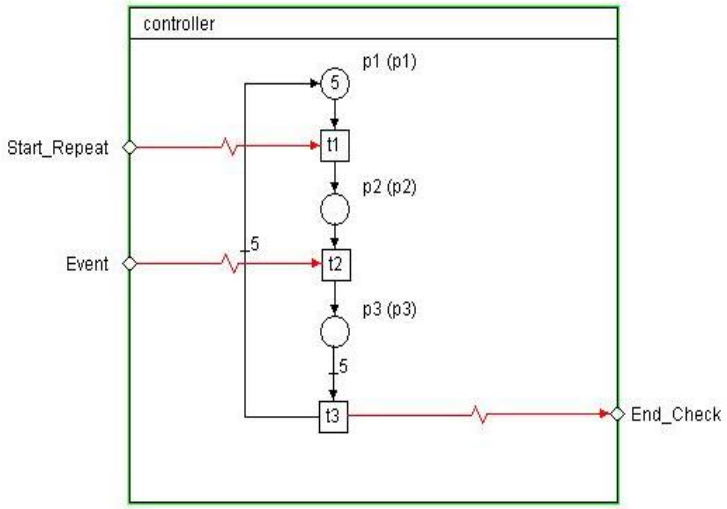


Figure 3-5 Repeat Module

4. RESULTS

This chapter discusses the proposed methodology to model complex event processing into formal languages. Section 4.1 briefly describes case study taken for this thesis. Section 4.2 explains modelling of case study in TNCES. Section 4.3 and Section 4.4 focuses TNCES models of different event processing languages statements and event pattern language statements respectively. Section 4.5 presents an example of an EPL statement and its translation into TNCES modules and then communication sequence between those modules. In the last, Section 4.6 emphasises on CTL formulae used to validate developed TNCES module.

4.1. Case Study

In order to model CEP, first events generated at Factory Floor are modelled in TNCES. For this purpose, Festo Modular Production System is selected as a case study.

Festo Modular Production System (MPS) system shown in Figure 4.1 is taken as a line case study in this thesis studies. The line consists of seven stations: Distribution, Testing, Processing, Handling, Robot, Assembly and Storing Station.

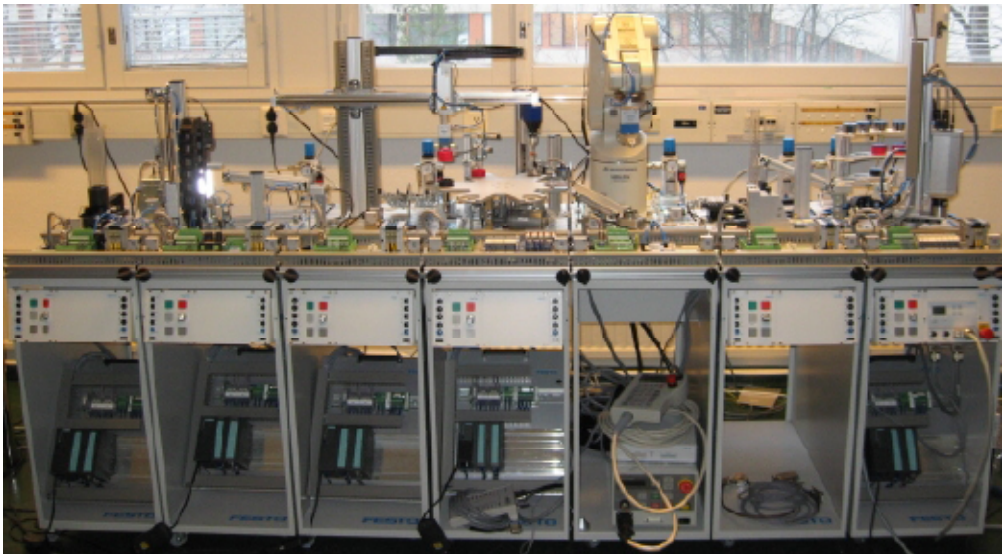


Figure 4-1 Line case study

The line is able to manufacture red, silver and black workpieces. Total manufacturing process consists of feeding workpieces by distribution station and then testing station checks colour and size of workpiece. The processing station again checks size of the workpiece and perform drilling. It also acts as a buffer. The handling station is used

to handle workpieces between stations and also to store workpieces. The next step is to perform assembly operations and store the magazines based on colour of workpieces and it is done by robot station. The assembly station supplies pistons, caps and magazines to assemble workpieces and assembly is done by robot. In the last step, storing station sorts the assembled workpieces according to its colour and stores them in appropriate place. Figure 4.2 shows production line from another view.

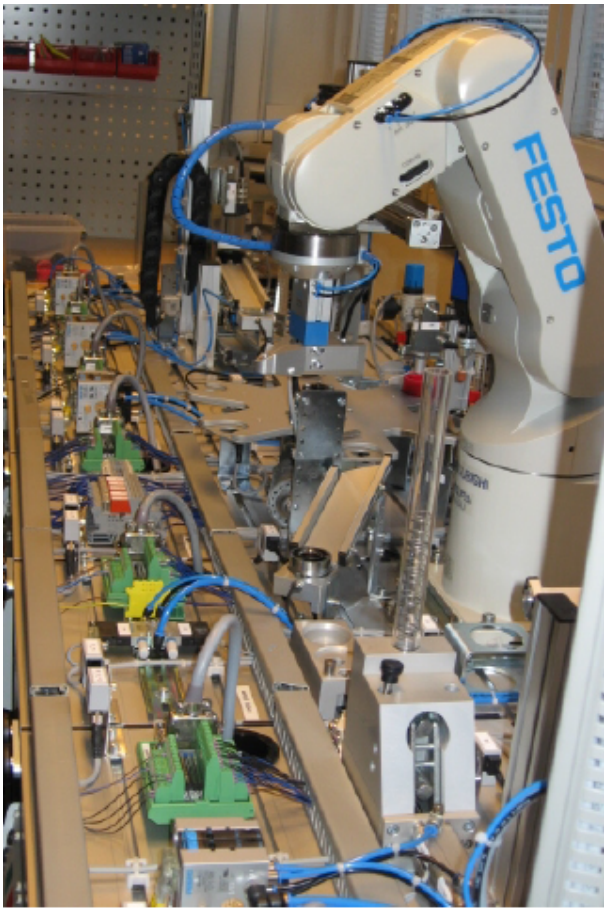


Figure 4-2 Line case study: Another view

4.2. Modelling Distribution Station

By considering that Festo MPS line is present at factory floor, distribution station of line is modelled in TNCES. The events generated at factory floor are taken from IPC standard IPC-2541 (IPC, ANSI 2001) and corresponding events at MES and ERP are also modelled in TNCES. The toolset used for supporting methodology proposed in this chapter is called Movida NCES Generator (MOVIDA).

4.2.1. Factory Floor

The Distribution System is the first equipment in the whole production line. It has got two zones, Stack Magazine and Changer Module.

Stack Magazine

It consists of following parts,

i. Double acting ejecting cylinder pushing workpiece out.
ii. Magazine Barrel of the stack magazine holding 8 workpieces.
iii. Through-beam sensor to detect available workpiece.
iv. Control valves for adjusting speed of ejecting cylinder.

Changer Module

This module has following parts,

i. Suction Cup to pick up the workpiece.
ii. Mechanical end stops to adjust swivelling range.
iii. Electric limit switches to sense end position.

Different events which are generated by Factory Floor are given below with their description.

Table 4-1 Events Generated at Factory Floor

Events	Description
ItemWorkStart	By Equipment to MES, after pressing of Start Button
ItemWorkPause	By Equipment to MES, after pressing of the Pause Button
ItemWorkResumed	By Equipment to MES, after resuming the work.
ItemWorkAbort	By Equipment to MES, if Stop Button is pressed
ItemTransferIn	**Stack Magazine** By Equipment to MES, when Magazine is in front position.
ItemTransferOut	**Changer Module** By Equipment to MES, when suction cup places the workpiece on the next station.
ItemTransferZone	By Equipment to MES, when workpiece changes zone from stack magazine to changer module.
ItemInformation	By Equipment to MES, when Bar Code of the item is read by Suction Cup
EquipmentBlocked	If there is already a workpiece on the next station. There is a signal from the next station, that there is already a work-piece.
EquipmentUnBlocked	There is no workpiece now on the next station.
EquipmentInitializationComplete	By Equipment, that equipment is being initialized.
EquipmentRecipeSelected	By Equipment, that Equipment Recipe sent by MES is selected
EquipmentRecipeReady	By Equipment, that selected recipe is ready for execution

EquipmentRecipeModified	By Equipment to MES, that equipment has stopped processing workpieces with last recipe and is ready to receive new recipe.
EquipmentError	By PLC, If any error occurs in the equipment like any human is near to the machine or robot collides with any machine or any zone of the equipment is not working properly.
EquipmentErrorCleared	By Equipment to MES, if error is cleared.
EquipmentInformation	By Equipment to MES and information of equipment is sent.
ProductionStarted	By Equipment to MES, after production has been started.
RecipeActive	By Equipment to MES that selected Recipe has been activated.
RecipeList	Recipe List is available.
RecipeListRequest	Request from Equipment to MES to send recipe list.
Refill_Workpieces	By 'Storage' module to 'Workpieces' module to refill workpieces to Storage.
WaitingforOperatorAction	By Equipment to MES, if there are no more workpieces in Stack Magazine, then machine halts and operator has to refill them and restart the machine.
WorkpiecesLessInNumber	By Equipment to MES that work pieces in Storage are less in number.
WorkpiecesShort	By 'Workpieces' module to 'Storage' module to inform that amount of workpieces in Stack Magazine is less than a specific amount.

The explanation of different modules modelled in TNCES and events generated by them on Factory Floor is given below.

Equipment Initialization

Equipment Initialization module makes sure that instruction to start a certain order from MES has been received and equipment is ready to process material using provided recipe by MES. It then informs MES about the commencement of the order.

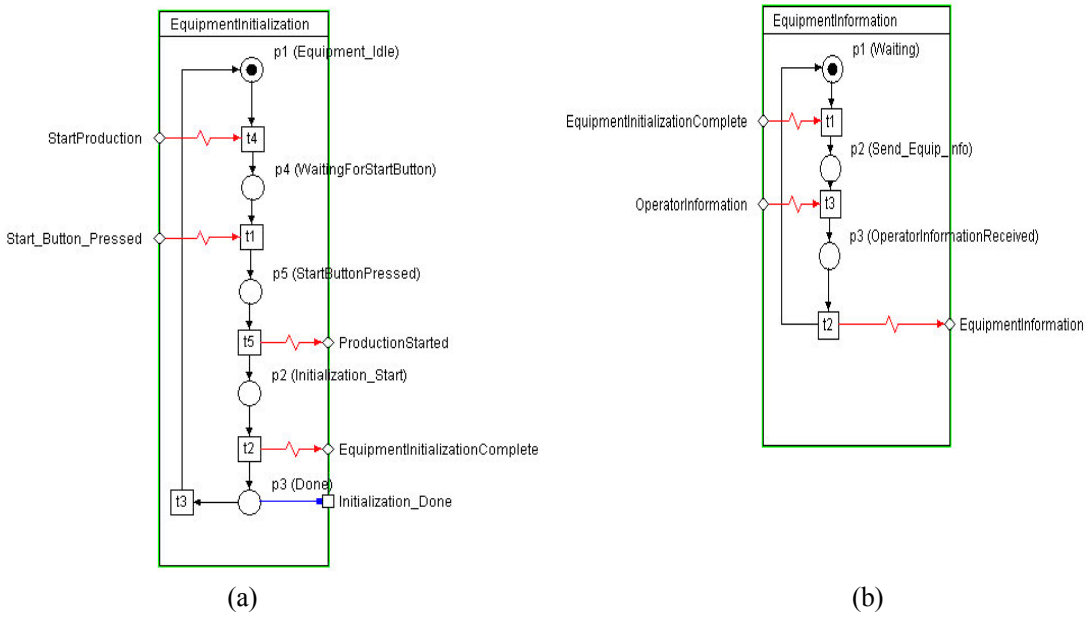
Figure 4.3 (a) shows the events linked to equipment initialization. Equipment receives an event 'StartProduction' from the MES and Operator present at shop floor presses the Start Button and equipment is initialized and it generates two events, 'ProductionStarted' and 'EquipmentInitializationCompleted'. 'ProductionStarted' event is sent to MES to inform that production of a certain order has been started and 'EquipmentInitializationCompleted' event is sent to next modules, 'RecipeRequest' and 'EquipmentInformation'.

This module is activated only when production is not being done on equipment before and there is a need to initialize the equipment.

Equipment Information

Equipment Information module sends the necessary information of equipment to MES e.g. number of lanes and zones in the equipment.

After getting an event 'EquipmentInitializationComplete' from Equipment initialization module and 'OperatorInformation' event from MES, an event 'EquipmentInformation' is sent to MES. Equipment Information module and its associated events is shown in Figure 4.3 (b).



(a) (b)

Figure 4-3 (a) Equipment Initialization Module (b) Equipment Information Module

Recipe Request

This module requests recipe list from the MES. The recipe list contains all recipes required in completion of an order. Figure 4.4 (a) displays Recipe Request module and all events corresponding to it.

If condition 'EquipmentInitializationComplete' is satisfied, 'RecipeRequest' module sends an event 'RecipeListRequest' to MES. MES acknowledges by sending an event 'RecipeListReceived' and 'RecipeRequest' module sends an event 'RecipeList' to next module 'Equipment_Recipe' and then informs back MES that sent recipe is active by sending an event 'RecipeActive'.

Equipment Recipe

TNCES model of this module is illustrated in Figure 4.4 (b). This module is responsible for making recipe operational and for taking necessary measures, if recipe is modified or changed by MES during the process.

After receiving an event 'RecipeDetails' which contains Recipe List from 'RecipeRequest' module, 'Equipement_Recipe' module sends an event 'EquipmentRecipeReady' to next module 'Workpiece_Flow' and condition 'Recipe_Ready' is enabled until MES decides to replace the recipe with the new one or modify the existing one. If new recipe is selected or existing recipe is modified, an event 'EquipmentRecipeModified' is sent to MES and 'Recipe_Ready' condition is not satisfied any more.

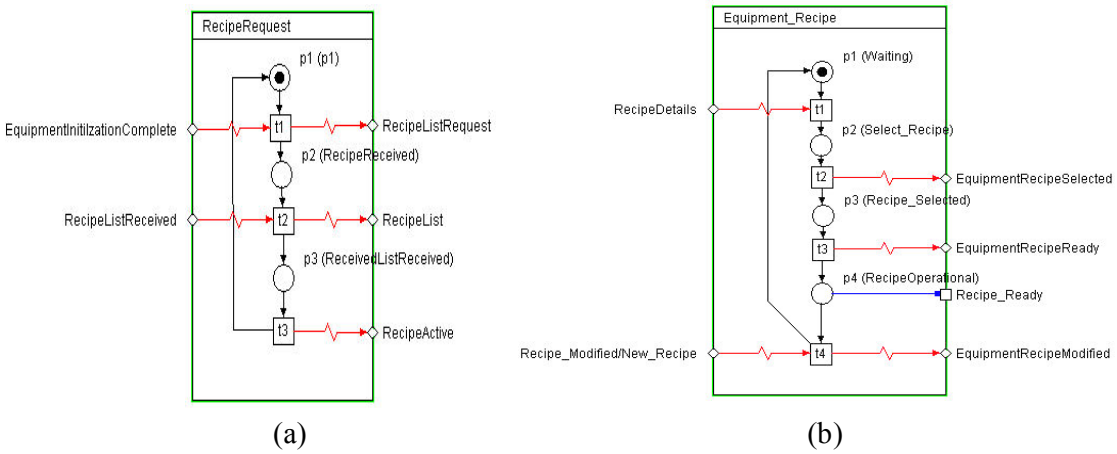


Figure 4-4 (a) Recipe Request Module (b) Equipment Recipe Module

Workpieces

Stack magazine in Distribution Station is able to store up to 8 workpieces. 'Workpieces' module makes sure that if there are enough workpieces in the stack magazine or not. If not, then this module sends necessary events to MES and 'Storage' module. Figure 4.5 shows the TNCES model of this module.

If 'Recipe_Ready' condition is satisfied which means that selected recipe is ready to be implemented and if there is a workpiece in the stack magazine, then an ejecting cylinder pushes a workpiece out of the stack magazine.

There are three possibilities on the basis of the workpieces in the Stack Magazine,

i. If remaining number of workpieces in Stack Magazine is four, then 'Workpieces' module sends an event 'WorkpiecesShort' to 'Storage' module. The 'Storage' module then immediately refills the stack magazine.

ii. If remaining number of workpieces in Stack Magazine is two, then an event 'EquipmentError' is sent to MES. MES instructs 'Storage' to refill the workpieces in Stack Magazine.

iii. For any reason, if Storage is unable to refill the workpieces and Stack Magazine goes empty, then FactoryFloor sends an event 'WaitingforOperatorAction' to MES and equipment stops. In this case, operator has to refill the workpieces by sending an event to 'Storage' manually.

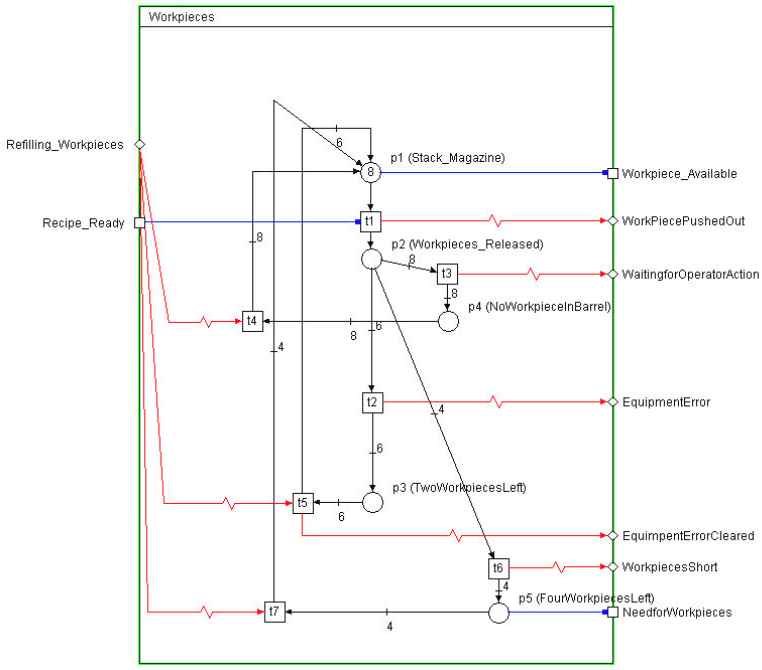


Figure 4-5 Workpieces Module

Storage

Storage is a part in factory floor which can accumulate up to 100 workpieces. Whenever, remaining quantity of workpieces in Stack Magazine reaches a specific amount, 'Storage' refills them. TNCES modelling of 'Storage' module is shown in Figure 4.6.

Let us suppose that 8 workpieces are refilled to Stack magazine already and as it can be seen in Figure 4.6 that remaining number of workpieces in place 'WorkpiecesInStorage' is 92. If number of workpieces refilled to Stack Magazine exceeds 50, then Storage sends an event 'WorkpiecesLessInNumber' to MES which means that workpieces in Storage are also less in number. MES responds back with event 'WorkpiecesAvailable' and refills the workpieces in the Storage.

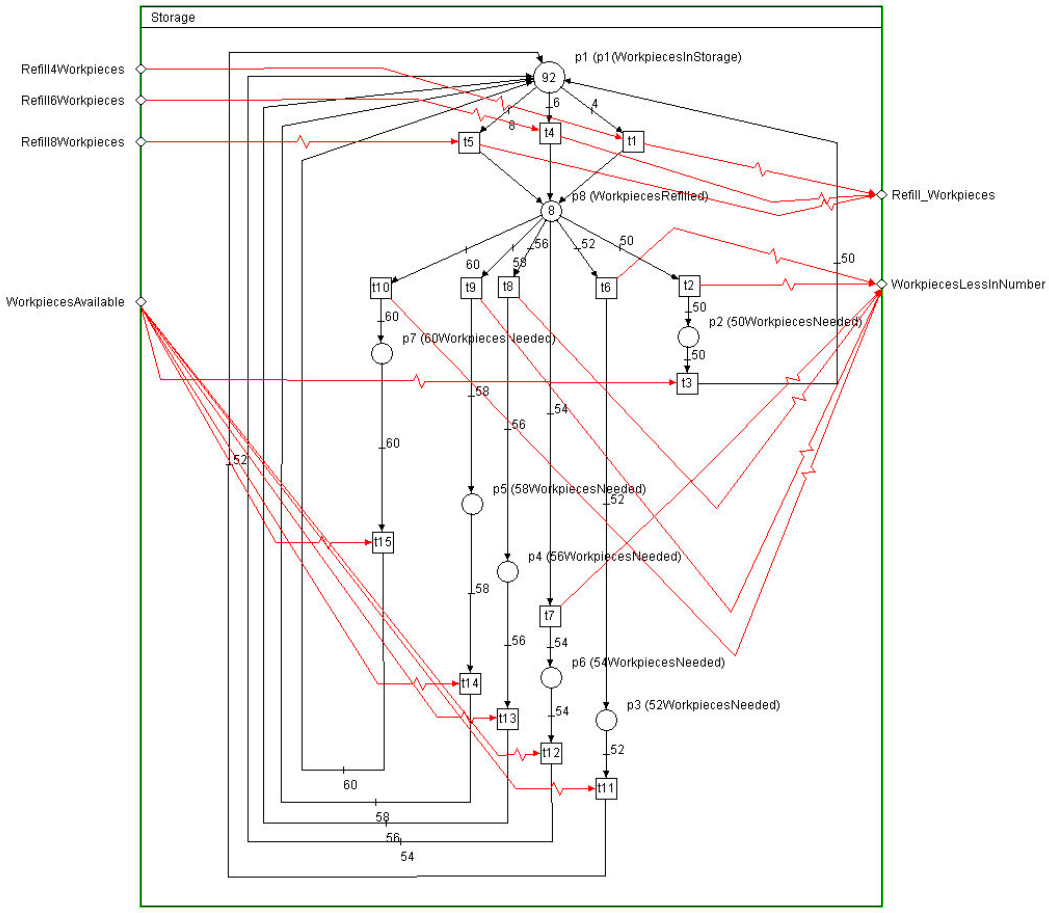


Figure 4-6 Storage Module

WorkPiece Flow

Figure 4.7 explains module 'WorkpieceFlow' modelled in TNCES.

In this module, following sequence takes place.

i. If workpiece is pushed out by ejecting cylinder out of stack magazine and recipe is active, then an event 'ItemWorkStart' is sent to MES:

ii. Then an event 'ItemTransferIn' is sent to MES explaining that workpiece is ready to be processed.

iii. Each workpiece has a unique bar code which is read by suction cup. This bar code contains information like lot number and order number to which this workpiece belongs. All this information is sent to MES through event 'ItemInformation'. By this event, MES can keep track of all the workpieces. MES can also determine which workpiece is under process and thus it can monitor the current status of the order.

iv. After workpiece has been pushed out of stack magazine and is picked by suction cup, an event named 'ItemTransferZone' is sent to MES indicating that workpiece has been transferred from one zone to another zone.

v. When workpiece moves to next station, event 'ItemTransferOut' is sent representing that workpiece has now transferred to next equipment.

If during the process, Pause button is pressed by an operator, event ‘ItemWork-Paused’ is sent to MES and whole equipment is paused. After pushing Resume button, equipment resumes working from same position from where it was paused and event ‘ItemWorkResumed’ is generated.

In case, operator stops whole equipment by pressing Stop button, equipment stops working and MES is informed by an event ‘ItemWorkAbort’. In this case, whole production has to be started from beginning by sending ‘StartProduction’ event from MES.

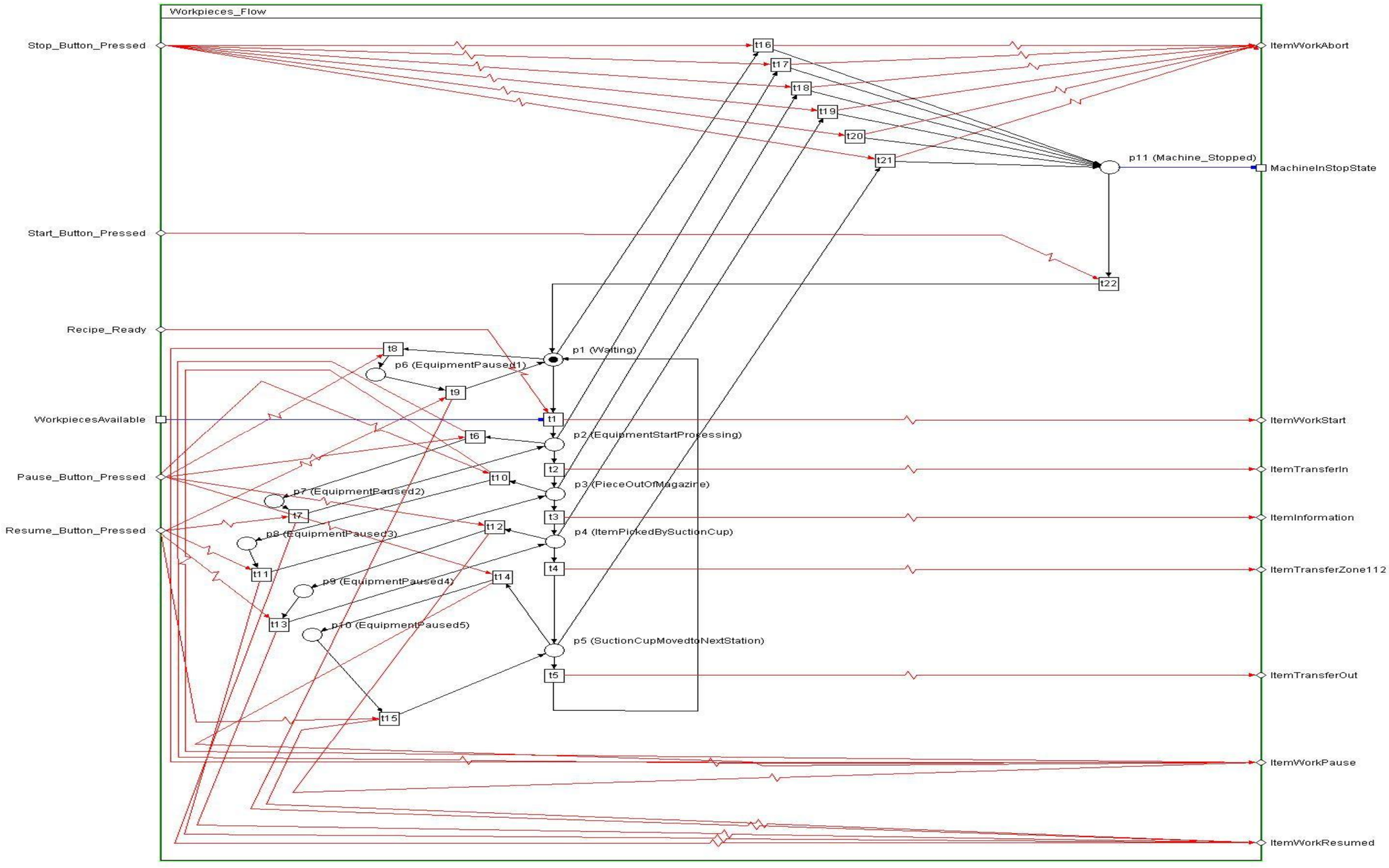


Figure 4-7 Workpiece Flow Module

4.2.2. Manufacturing Execution System (MES)

Description of different events generated on MES is given in Table 4.2.

Table 4-2 Events generared at MES

Events	Description
BOM	By MES to ERP, to inform workpieces in Storage of Factory Floor are less in amount and to send new workpieces.
EquipmentStopped	By MES to ERP, to inform Equipment is not under production.
NewWorkpiecesAvailable	By MES to Factory Floor, informing new workpieces are refilled to Storage.
PauseTime	By MES to ERP, notifying how many hours, machine was paused.
Production_Started	By MES to ERP, reporting production by equipment has been started.
OperatorInformation	By MES to Factory Floor, giving information about new operator.
RecipeDetails	By MES to Factory Floor, in response of 'RecipeListRequest'.
RecipeModified/NewRecipe	By MES to Factory Floor, when MES decides to set up new recipe or modify existing recipe.
RefillingWorkpieces	By MES to 'Storage' module of Factory Floor in response of 'OperatorActionNeeded', instructing to refill 8 workpieces to Stack Magazine.
Start_Production	By MES to Factory Floor, when MES wants to start production.
StoppageTime	By MES to ERP, notifying how many hours, machine was not in stop mode.

Different modules present in MES and events generated by them are given below.

Start Production

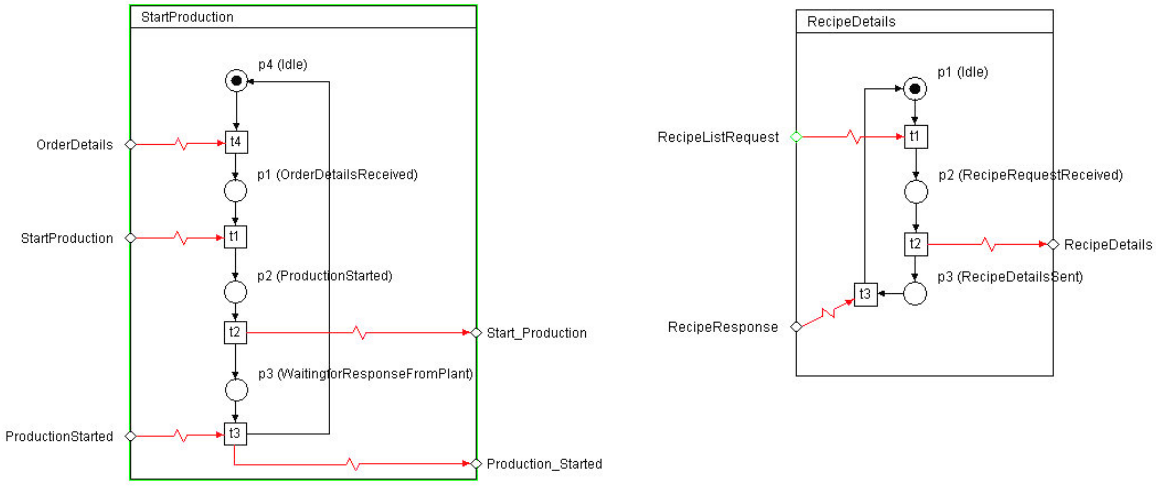
Start Production module is used by MES to schedule and start the production.

ERP informs MES about any new order by an event 'OrderDetails' and MES depending on its capacity schedule it. After planning, MES sends an event 'Start_Production' to Factory Floor and Operator present on the Factory Floor presses the Start button and equipment sends back the event 'ProductionStarted' by which MES comes to know that production has been started. MES also informs ERP about the starting of the production by which ERP can monitor the time of the completion of the order. After sending event 'ProductionStarted', equipment present on Factory Floor starts to initialize and so on. TNCES model of this module is shown in Figure 4.8 (a) below.

Recipe Details

This module sends recipe details to equipment when requested. Figure 4.8 (b) shows its TNCES model.

Equipment sends an event 'RecipeListRequest' to MES and MES sends then all the details of the recipe list and including recipes by an event 'RecipeDetails'. After activating sent recipe, equipment generates an event 'RecipeActive' which is received by MES as an event 'RecipeResponse'.



(a) (b)

Figure 4-8 (a) Start Production Module (b) Recipe Details Module

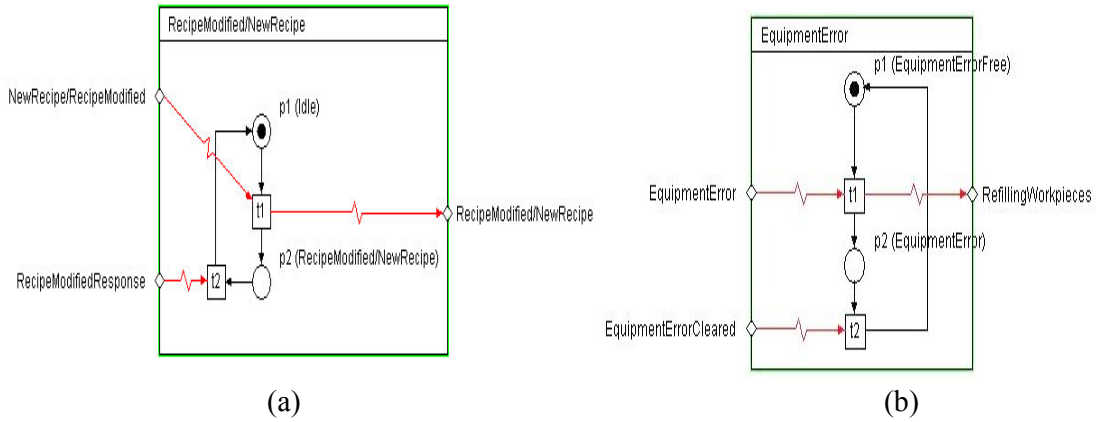
Recipe Modified/New Recipe

Figure 4.9 (a) shows TNCES model of this module. If MES decides to modify the recipe being processed or to employ new recipe, then MES sends an event 'RecipeModified/NewRecipe' to Factory Floor. Factory Floor after getting this event stops processing new workpieces and requests details of new recipe by sending 'RecipeModifiedResponse'. This event is received by 'RecipeModified/NewRecipe' module by which MES comes to know that recipe has been changed. Another module which receives this event is 'RecipeDetails' as 'RecipeListRequest' event. After that, MES sends details of new recipe to equipment.

Equipment Error

Equipment error event arises in Factory Floor whenever there are two workpieces left in stack magazine. Figure 4.9 (b) demonstrates working of module 'EquipmentError'.

After getting this event, MES sends an event 'RefillingWorkpieces' to Storage which in response refills six workpieces to stack magazine. Factory floor then sends an event 'EquipmentErrorCleared' back to MES showing that equipment is free of any error.



(a) (b)

Figure 4-9 (a) Recipe Modified/New Recipe Module (b) Equipment Error Module

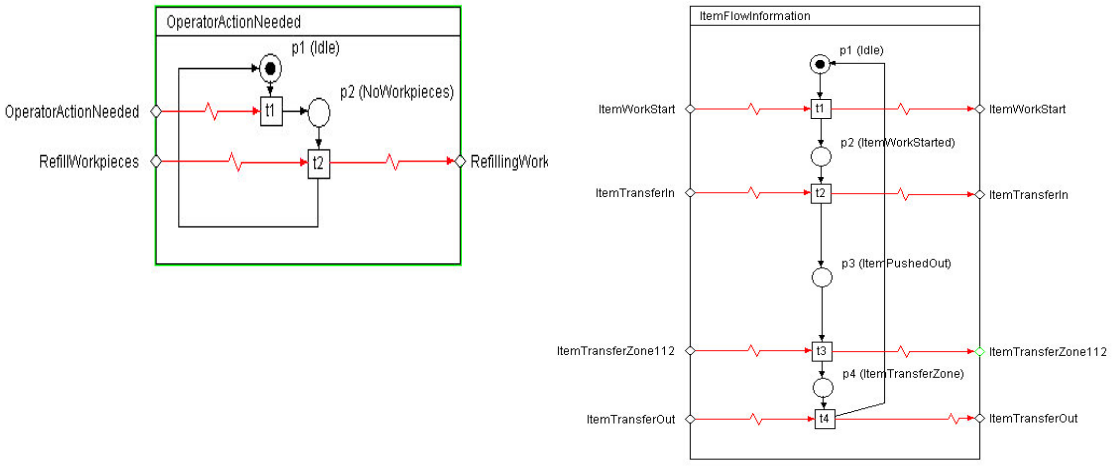
Operator Action Needed

This module works whenever, equipment is in such a condition that further processing is not possible and involvement of operator is necessary to eradicate such condition. In case of Distribution System, if there is not workpieces left in the stack magazine, then Factory Floor sends an event 'OperatorActionNeeded' to MES and MES has to manually instruct Storage to refill eight workpieces to Stack Magazine by sending an event 'RefillingWorkpieces'. Figure 4.10 (a) demonstrates of working of this module.

Item Flow Information

This module keeps track of the workpieces under production e.g. location of a workpiece belonging to any certain order in Distribution Station. Its TNCES model is exhibited in Figure 4.10 (b).

The events in this module are same as of 'Workpiece Flow' module in Factory Floor except 'ItemWorkAbort', 'ItemWorkPause' and 'ItemWorkResumed' events. So, whenever workpiece changes its location in equipment present on Factory Floor, it informs MES and in this way, MES gets to know the exact location of any workpiece.



(a) (b)

Figure 4-10 (a) Operator Action Needed Module (b) Item Information Module

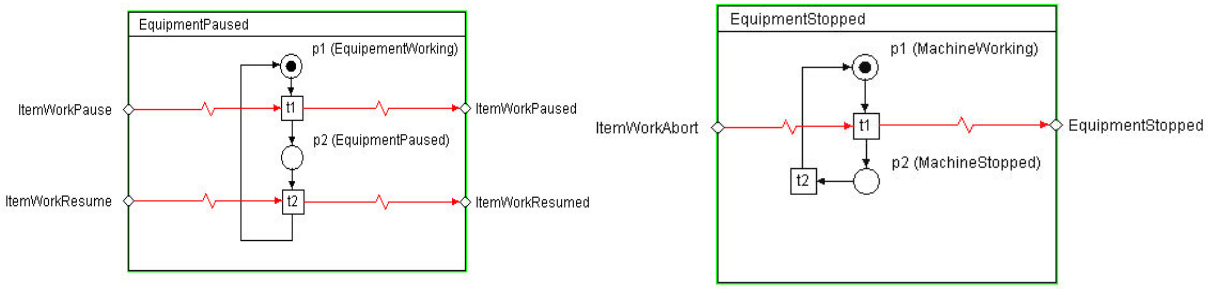
Equipment Paused

If operator pauses the equipment for any reason, then equipment sends an event 'ItemWorkPause' to MES. As soon as equipment is paused, counter starts and after every 60 minutes of total pausing time of equipment, an event 'PauseTime' is send to ERP. After recommencing working of equipment, MES receives 'ItemWorkPause'. If during any time of processing, equipment is stopped, then this module comes to its initial state.

In this way, ERP can show its concern to MES about delaying of order due to frequent pausing of machines and MES in turn can inquire operator about the reasons of pausing the equipment and can try to rectify it. Following these lines, MES can reduce the downtime of equipment. Figure 4.11 (a) shows all the conditions and events generated in this module.

Equipment Stopped

If for any reason, working of equipment is terminated, then equipment informs immediately by an event 'ItemWorkAbort'. In the similar fashion as in 'Equipment Paused' module, after every 60 minutes of total stoppage time of equipment, MES informs ERP by an event 'StoppageTime'. Figure 4.11 (b) shows working of 'EquipmentStopped' module.



(a) (b)

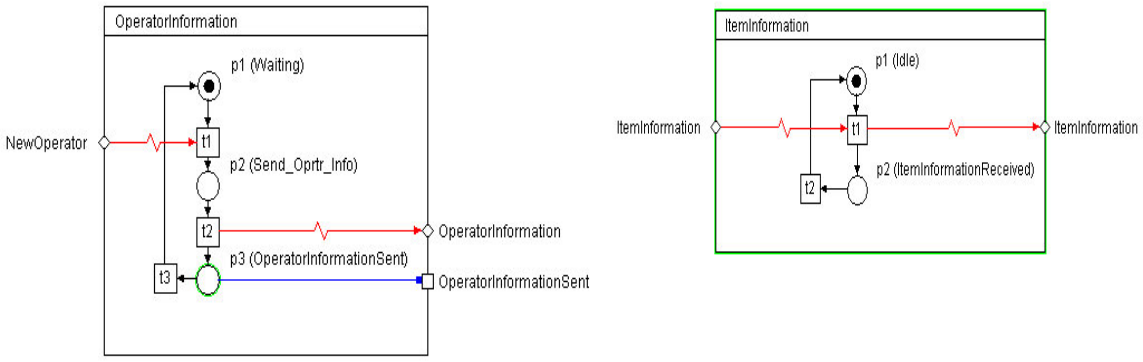
Figure 4-11 (a) Equipment Paused Module (b) Equipment Stopped Module

Operator Information

Whenever, any new operator starts his/her shift, MES sends his/her information to equipment via event named 'OperatorInformation' as shown in Figure 4.12 (a).

Item Information

TNCES model of 'Item Information' module is shown in Figure 4.12 (b). This module consists of an event generated from Factory Floor containing all the information in the bar code of the workpiece e.g. unique number of the workpiece and the order to which it belongs.



(a) (b)

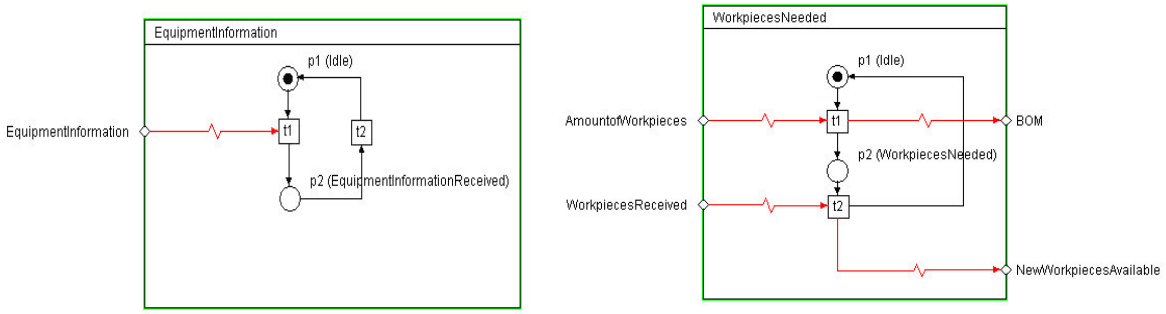
Figure 4-12 (a) Operator Information Module (b) Item Information Module

Equipment Information

This module is responsible for receiving information related to equipment e.g. number of zones, its software etc and is shown in Figure 4.13 (a).

Workpieces Needed

This module shown in Figure 4.13 (b) is approached whenever amount of workpieces in the Storage of the Factory Floor is less than 50. This module 'WorkpiecesNeeded' on getting an event 'AmountofWorkpieces', sends BOM (Bill of Materials) to ERP with the help of an event 'BOM' demanding for new workpieces. When new workpieces are provided by ERP, MES sends an event 'NewWorkpiecesAvailabe' to Storage.



(a) (b)

Figure 4-13 (a) Equipment Information Module (b) Workpiece Needed Module

4.2.3. Enterprise Resource Planning (ERP)

Events which are generated on ERP with their brief detail are given in Table 4.3.

Table 4-3 Events generated at ERP

Events	Description
OrderDetails	By ERP to MES, upon receiving new order ERP sends its detail.
NewWorkpiecesAvailable	By ERP to MES, that workpieces are received from supplier.

Different modules and their generated events in ERP are described below.

Production Tracking

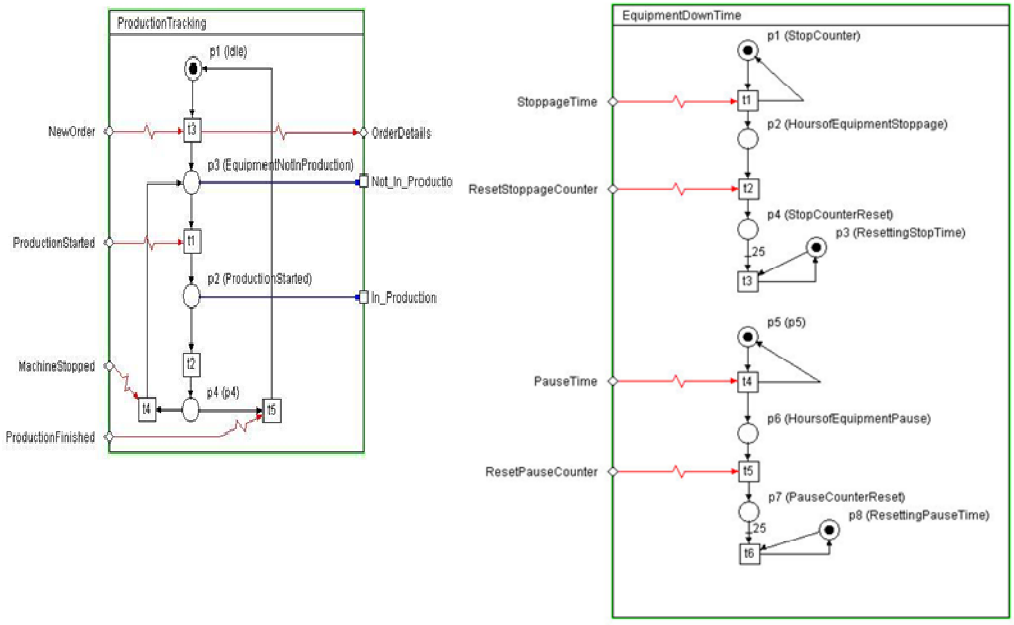
The module 'Production Tracking' monitors either equipment is under any production or not. When ERP receives new order, it sends detailed explanation to MES by sending an event 'OrderDetails'. And when production starts at Factory Floor, ERP is informed by an event 'ProductionStarted'. If production is aborted in between, ERP is updated by an event 'MachineStopped'. On receiving event "ProductionFinished" from last station of line, token returns back to "Idle" place. Figure 4.14 (a) elaborates functioning of this module.

Equipment Downtime

Total downtime of the equipment is informed to ERP, from which ERP determines the efficiency and reproach MES if completion of order is being delayed.

After equipment being stopped or paused for total 60 minutes, MES informs ERP by events called 'StoppageTime' and 'PauseTime'. In this fashion, ERP can find out how many hours, equipment was not in working condition.

Counters can be reset using events 'ResetStoppageCounter' or 'ResetPauseCounter'. Conditions and events related to this module are portrayed in Figure 4.14 (b).



(a) (b)

Figure 4-14 (a) Production Module (b) Equipment Down Time Module

Bill of Materials (BOM)

A *bill of materials (BOM)* is a list of the raw materials, sub-assemblies, components or parts and the quantities of each needed to manufacture an end product.

Whenever, new workpieces are needed at Factory Floor, Distribution System informs MES and MES informs ERP through an event 'BOM'. ERP on getting this event sends Purchase Order (PO) to corresponding supplier. When supplier sends required number of workpieces, ERP

informs MES by an event 'NewWorkpiecesAvailable' and in turn MES informs Storage of Distribution System. The module "BillOfMaterials (BOM)" is shown in Figure 4.15.

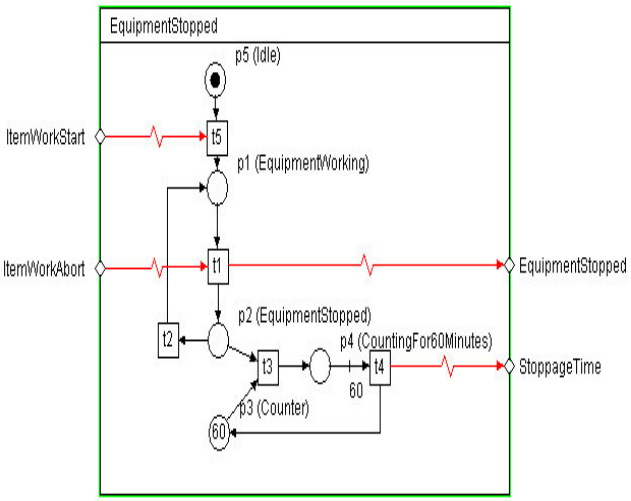


Figure 4-15 Equipment Stopped Module

4.3. Modelling Event Processing Language Statements

This section describes two examples having simple EPL constructs represented in TNCES.

4.3.1. Example 1: AutoID RFID Reader

This example is taken from Esper (ESPER 2011) in which there is an array of 4 RFID Readers with Ids, urn:epc:1:4.16.30, urn:epc:1:4.16.32, urn:epc:1:4.16.36 and urn:epc:1:4.16.38 which generates XML documents after sensing RFID tags as pallets come within the range of one of the readers. The generated XML document contains information such as reader sensor ID, observation time and tags observed. An EPL statement computes the total number of tags per reader sensor within last 60 seconds.

The EPL statement which computes the total number of tags per sensor reader is given below,

```
select ID as sensorId, sum(countTags) as numTagsPerSen-
                                sor
      from AutoIdRFIDExample.win:time(60 seconds)
 where Observation[0].Command = 'READ_PALLET_TAGS_ONLY'
                          group by ID
```

Figure 4.16 shows snapshot of AutoID RFID Receiver. User has to give number of events to be generated and given number of events is generated. The generated event contains information time of observation, SensorID and total number of tags generated.

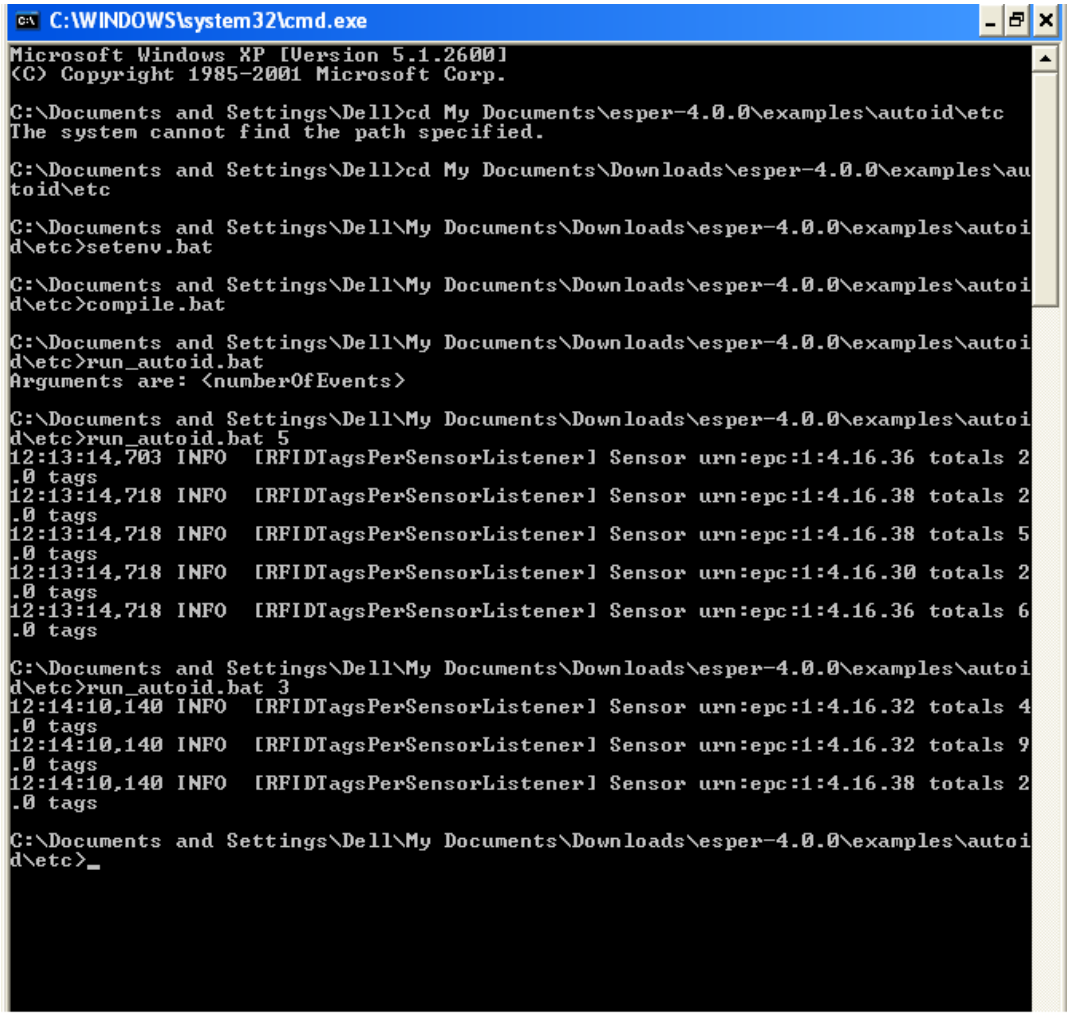


Figure 4-16 AutoID RFID Reader

Let's consider that there are two sensors R1 and R2 installed in Factory Floor and they generate an event stream AutoIdRFIDExample whenever they read any pallet having a RFID tag. We want to determine IDs of the tags and total number of tags per ID counted within 60 seconds. The EPL statement for this example is

```
select ID, sum(countTags)
from AutoIdRFIDExample.win:time(60 seconds)
group by ID
```

The TNCES representation of this example is given in the figure below. The TNCES model consists of Select, From, Properties, EventStreams, AutoIDRFIDExample Engine and ID+Sum modules. The Select module initiates the EPL statement followed by From module which specifies name of event stream from which events are selected. In the EventStreams module, desired event stream is selected. The AutoIDRFIDExample Engine captures events from Factory Floor and ID+Sum module selects IDs of tags read by sensors and sum of the total tags within 60 seconds.

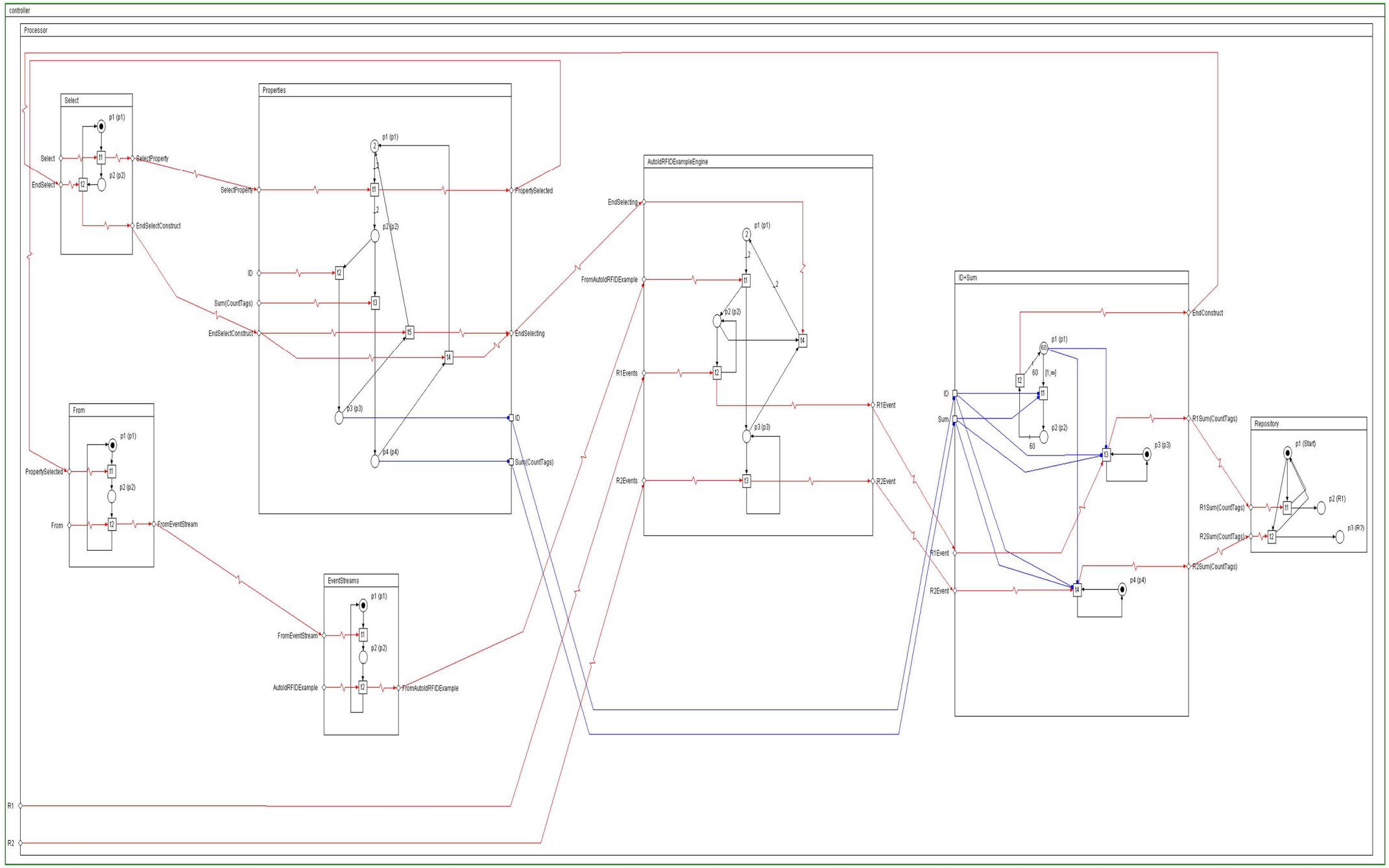


Figure 4-17 TNCES model of Example

4.3.2. Example 2

Consider another example which is having an event stream called EStream1 and it comprises of three types of events E1, E2 and E3. If we want to select all events plus E1, E2 and E3 events separately, then EPL statement for this purpose is,

```
select *,E1,E2,E3 from EStream1
```

In the Figure 4.18, the EPL statement given above is translated in TNCES. There are 5 modules namely Select, From, EventStreams, Properties and EStreamEngine. The Select module starts executing the EPL statement. The EventStreams module contains list of all events streams and From module selects desired event stream from that list. Properties module signifies properties of the events to be selected and the EStreamEngine module selects all indicated properties in the EPL statement which are all, E1, E2 and E3 in this example.

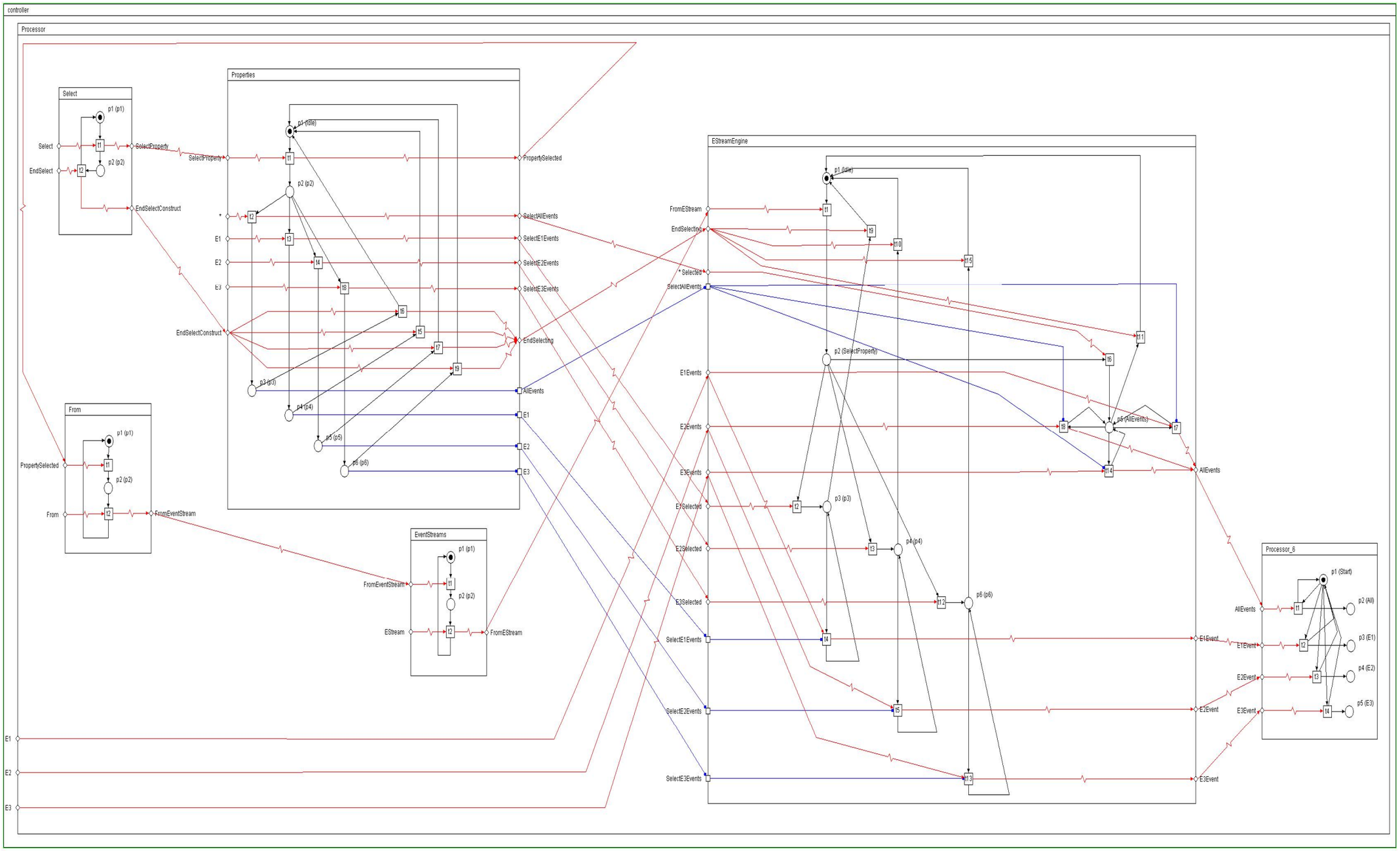


Figure 4-18 TNCES model of Example 2

4.4. Modelling Event Pattern Language Statements

In this section, different event pattern language statements are represented in TNCES. To support developed models in Section 3.4, an example is given in which event pattern language statement is translated in TNCES.

4.4.1. Example

Let us consider a production line which produces workpieces of 3 colours i.e. red, black and silver. Following pattern matches any red, black or silver workpiece is manufactured within 60 seconds.

```
every (red) or every (black) or every (silver) where
                timer:within(60 seconds)
```

The above pattern is represented in TNCES in Figure 4.19. There are 3 separate Every module for red, black and silver workpieces. Then there is an OR module which selects any of the workpiece manufactured. The timer:within module keeps check that pattern remains true only for 60 seconds and hence events are generated only for workpieces manufactured within 60 seconds.

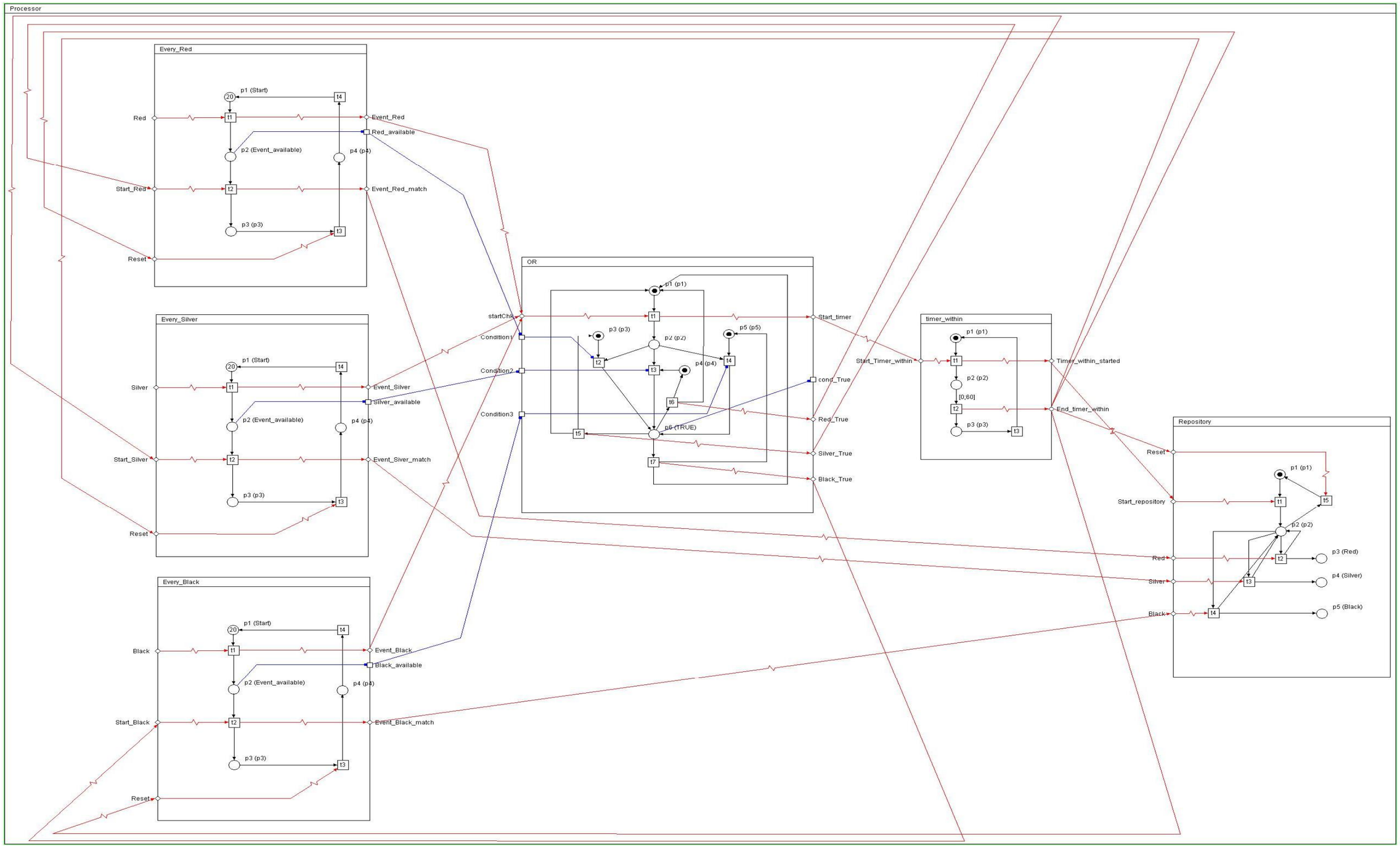


Figure 4-19 TNCES module of Example

4.5. TNCES Modelling of an EPL and Pattern statement

Let us consider a conveyor system having two pallets palletA and palletB on it. EventA and EventB are generated on passing of palletA and palletB from the specific location on conveyor respectively. Following EPL statement counts number of tags whenever a specific pattern of EventA followed by EventB is matched.

```
select count(Tags)
from pattern [every (EventA → EventB)]
```

The above statement is translated into TNCES by developing different modules for different purposes which are described below.

- Controller module – This module activates all modules on their turn one by one by sending events.
- Select module – Select module initiates the Select construct.
- Properties Module – This module has list of all properties and user can select desired property by sending respective event. In our case, we want to select Count of Tags, so a Count event is sent to "Properties" module.
- From Module – From module has list of all event streams or patterns from where we want to select events or event properties.
- Pattern Module – Pattern module has sub-modules needed to represent specific pattern. In this case, we need "FollowedBy" and "Every" modules to represent "every (EventA→ EventB)".
- FollowedBy Module – This module makes sure that EventA is captured first and then EventB is captured.
- EveryA and EveryB Modules – These modules are responsible for capturing and matching events EventA and EventB.
- Count Module – Count module counts number of tags. Similarly, every property has its own dedicated module.

The sequence diagram shows order of interaction between different modules in Figure 4.20. First "Controller" sends an event "Select" to module "Select". The "Select" module sends an event "SelectProperty" to module "Property". The module "Property" has the list of properties from where desired property can be selected. Each property in the EPL statement has its own module but we need to trigger only needed module(s) which is "Count" in our case. The "Controller" module specifies that property called "Count" of the events has to be selected by sending event "Count" to "Properties" module. The "Properties" module in turn sends an event "Count" to "Count" module in order to activate it.

After property is selected, "Properties" module sends an event "PropertySelected" to "From" module which is an indication that "From" module can start working. "From" module has list of all possible event streams. After "Controller" module sends an event

"From" to "From" module, "From" module specifies event stream or pattern from where we are going to select events by sending an event "FromPattern".

The "Controller" module signifies that its turn for "Pattern" module to start working by sending an event "Pattern1". The "Pattern" module starts functioning after sub-module "FollowedBy" generates an event "Start_Every_A" to sub-module "EveryA". After EventA is matched, sub-module "EveryA" sends back event "Event_A_match". Then "FollowedBy" sub-module sends an event "Event_B_match" to sub-module "EveryB" so that EventB can be matched. After EventB is matched, "Event_B_match" is sent back to sub-module "FollowedBy" indicating whole pattern is matched and complex event can be generated. As a result, sub-module "FollowedBy" generates Complex Event and sends relevant event "ComplexEvent" to module "Count". The module "Count" keeps on counting number of tags and sends an event "EndConstruct" to module "Controller" after EPL statement stops executing.

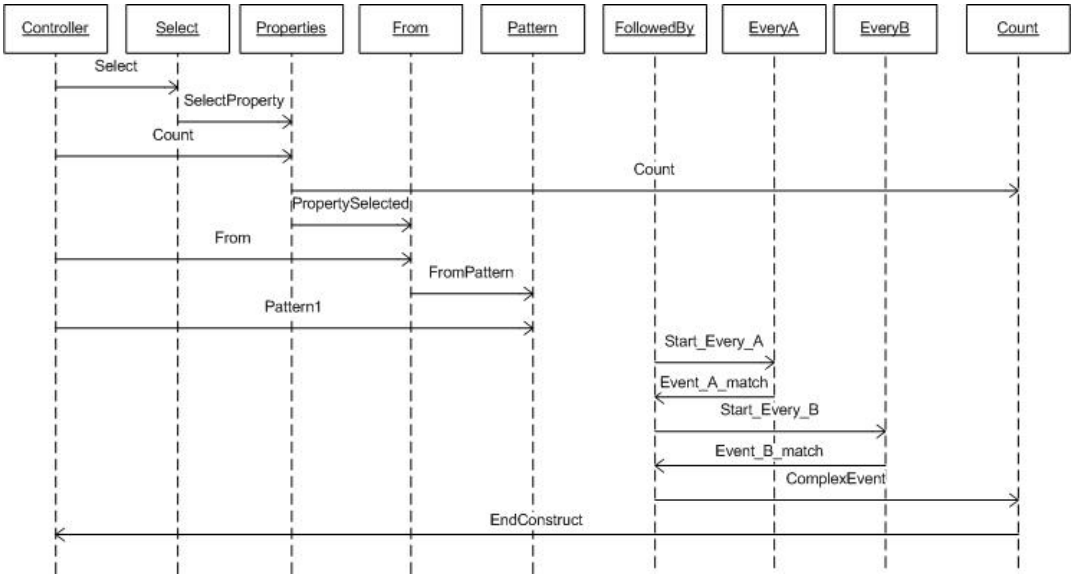


Figure 4-20 Sequence Diagram for EPL statement

Complete TNCES model is displayed in Figure 4.21.

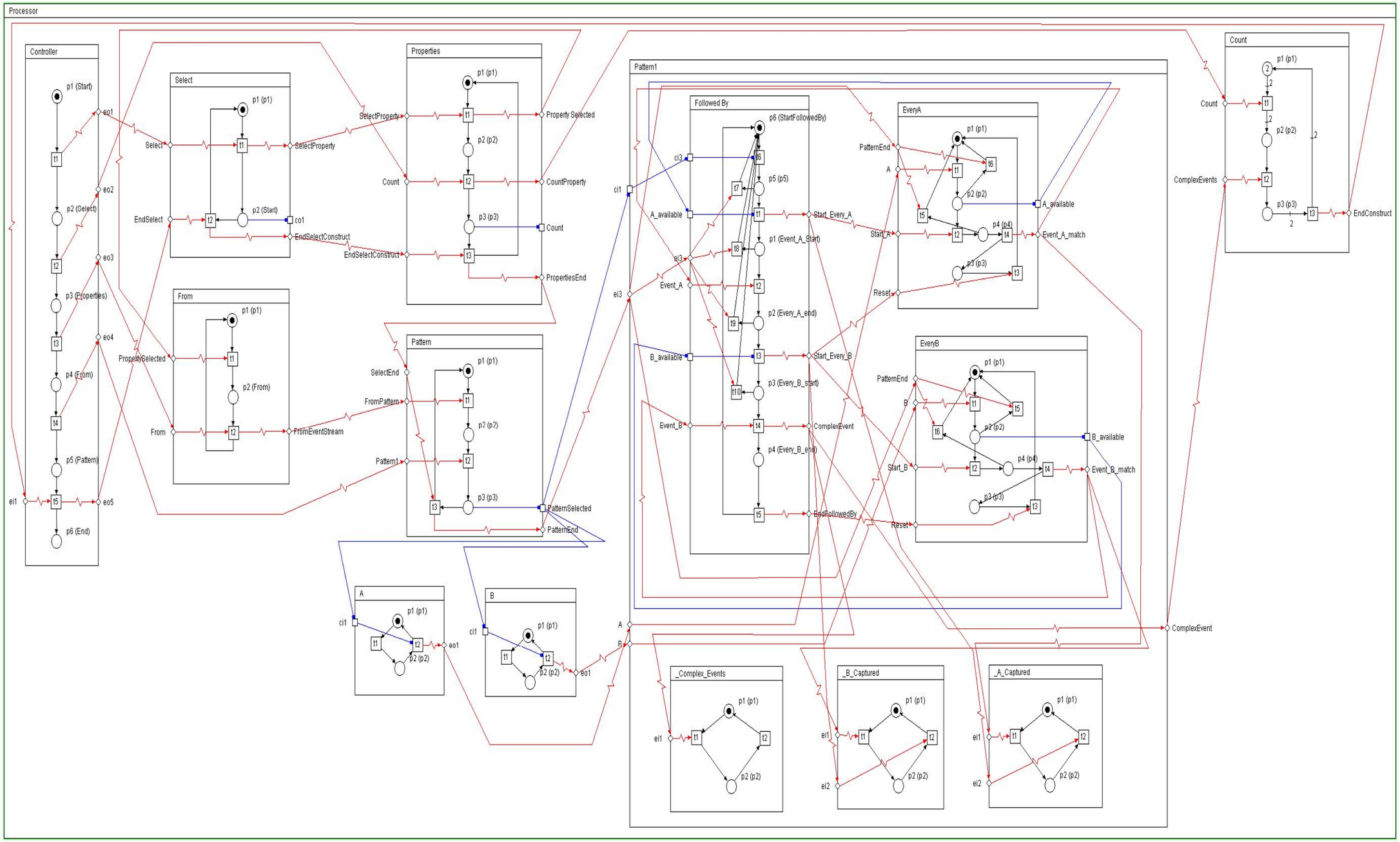


Figure 4-21 TNCES module of EPL and Pattern Statement

4.6. CTL Formulae for Validation

CTL formulae used to validate the TNCES system given in Figure 4.26 are given below in Table 4.4 along with their brief description. In Fig. 4.21, m(p29), m(p30), m(p27), m(p28), m(p25) and m(p26) denote those places in Table 4.4 which receive a token when complex event is not yet generated, when complex event is generated, when EventB is not yet matched, when EventB is matched, when EventA is not yet matched and when EventA is matched respectively. From the table below, we can see that our designed model is working as intended.

Table 4-4 CTL Formulae

Req. No.	Requirement	Corresponding CTL Formula	Result	Description
1.	There exists a path that does not generate Complex Event until EventA and EventB is captured.	E[(m(p29)=1) U (m(p28)=1 AND m(p26)=1)]	True	This formula checks whether modelled system generates a complex event only after capturing both events EventA and Event B. The result is true.
Counter 1	There exists a path that generates Complex Event until EventA and EventB is captured.	E[(m(p30)=1) U (m(p28)=1 AND m(p26)=1)]	False	This formula is converse formula of Requirement 1 and it turns out to be false.
2	There exists a path that does not capture EventB until EventA is captured.	E[(m(p27)=1) U (m(p26)=1)]	True	This formula checks if modelled system every time first captures EventA and then EventB and the result is true.
Counter 2	There exists a path that captures EventB until EventA is captured.	E[(m(p28)=1) U (m(p26)=1)]	False	This formula is opposite of Requirement 2 and it checks if there is any path which captures EventB before EventA and it results in false.
3	There exists a path that does not capture EventA until EventB is captured.	E[(m(p25)=1) U (m(p28)=1)]	False	This formula ensures if there is a possibility to capture EventB before EventA and outcome is false.
Counter 3	There exists a path that captures EventA until EventB is not captured.	E[(m(p26)=1) U (m(p27)=1)]	True	This formula is counter formula for Requirement 3 and it checks if every time EventA is captured

				before EventB. The result turned out to be true
4	There exists a path that generates Complex Event until EventA is captured.	E[(m(p30)=1) U (m(p26)=1)]	False	This formula validates if there is any chance that Complex Event is generated if only EventA is captured and result is false.
5	There exists a path that generates Complex Event until EventB is captured.	E[(m(p30)=1) U (m(p28)=1)]	False	This formula checks if there is any path which generates Complex Event if only EventB is captured and result is false.

5. CONCLUSIONS AND FUTURE WORK

5.1. Contributions

Despite of remarkable progress in the field of complex event processing, yet compact set of rules and standards for CEP are needed to represent real time events in virtual world. (Dindar, N., Balkesen, C. ; Kromwijk, K., Tatbul, N. 2009) (Kumar, S. 2009) (Hemani, A., Shamsi, J. 2010) (Leavitt, N. 2009).

By translating complex event processing into TNCES, we have defined set of standards and rules for processing complex events. TNCES also allows us to prevent different uncertainties involved in an event driven architecture EDA like machine breakdown and deadlocks. Also planning and scheduling EDA is more governed conveniently. Additional advantage of proposed methodology is insertion and deletion of elements to/from the model.

Furthermore, the designed methodology of formal modelling CEP systems also opens the path to the validation of events. It can be verified beforehand whether we have designed the correct model of the system or not. Also, we can validate if certain properties are held by our system or not.

5.2. Lessons Learned

This thesis includes research areas like Formal Methods, Complex Event Processing, Cognitive Science, and Temporal Logics.

5.3. Future Research Directions

Very comprehensive research has been done on the development of multi agent systems MAS in past decade, but on the other hand, small amount of work is done on formal modelling methods of MAS. (Zhenhua Yu, Zhiwu Li 2005)

Therefore, future trends could include multi agents which on the basis of different events can take real time actions. For example, if agent receives an event “Equipment-Blocked”, it can operate to take equipment of this situation. Also, agents could also help to resolve different legacy problems in manufacturing systems and integrate with CEP based systems in different activities like designing, planning, scheduling and execution.

Moreover, nowadays first system is modelled in TNCES and then validation techniques are applied on the system. But formal models do not include knowledge about sensors and actuators. If we incorporate knowledge of sensors and actuators in the

model, then we can introduce significant fairness constraints for the verification step of the model.

Finally, adding ontologies could further support accurate development of models at run time by recognizing suitable recovery strategies for equipment.

REFERENCES

Adrian Paschke, “Design Patterns for Complex Event Processing”, In Proceedings of 2nd International Conference on Distributed Event-Based Systems (DEBS'08), Rome, Italy, 2008.

Adrian Paschke, “A Semantic Design Pattern Language for Complex Event Processing”, AAAI Spring Symposium, 2009.

Ahmed Hambaba, “Soft-Object Technology for Flexible Machining Systems (FMS)”, In Proceedings of Second International Conference on Intelligent Processing and Manufacturing of Materials, Honolulu, USA, July 1999.

Banaszak Z.A, Krogh B.H., “Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows”, IEEE Transactions on Robotics and Automation, 1990

Baoan Li and Minxing Li, “Research and Design on the Refinery ERP and EERP Based on SOA and the Component Oriented Technology”, In Proceedings of International Conference on Netwrokign and Digital Society, Guizhou, China, May 2009.

Chuanzhen Zang, Yushun Fan, “Complex event processing in Enterprise Information Systems based on RFID”, 2007.

Clarke E.M, Grumberg O., Peled D.A., “Model Checking”, MIT Press, 2001, ISBN 0262032708 9780262032704

Curl, A., Fertalj, K., “A review of enterprise IT integration methods”, In Proceedings of 31st International Conference on Information Technology Interfaces, 2009.

David C. Luckham and Brian Frasca, “Complex Event Processing in Distributed Systems”, Stanford University, August 2008.

Descrochers A., “Modeling and control of automated manufacturing systems”, IEEE Press, 1989, ISBN:0-8186-8916-1, pp.239-251

Dindar, N. ; Balkesen, C. ; Kromwijk, K. ; Tatbul, N., “Event Processing Support for Cross-Reality Environments”, In Proceedings of IEEE Pervasive Computing, Zurich, 2009

Fanti M. P., Zhou M.C., “Deadlock Control Methods in Automated Manufacturing Systems”, IEEE Transactions on Systems, Man and Cybernetics, 2004

Fevzi Belli and Karl-E Grosspietsch, “Specification of Fault-Tolerant System Issues by Predicate/Transition Nets and Regular Expressions-Approach and Case Study”, In Proceedings of IEEE Transactions on Software Engineering,, Vol. 17, No. 6, June 1991

Hanisch, H-M., Thieme, J., Luder, A. and Wienhold, A., “Modelling of PLC behaviour by means of timed net condition/event systems”, paper presented at the 6th International Conference on Emerging Technologies and Factory Automation, pp.391–396, 1997

Heinz Roth, Josef Schiefer, Hannes Obweger and Szaboles, “Event Data Warehousing for Complex Event Processing”, In Proceedings of Fourth International Conference on Research Challenges in Information Science, France, May 2010

Hemani, A.; Shamsi, J., “Foundations of a generic design for complex event processing” International Conference on Information and Emerging Technologies (ICIET), 2010

Hyun Joong Yoon, Doo Yong Lee, “Deadlock-free scheduling method for track systems in semiconductor fabrication”, IEEE International Conference on Systems, Man, and Cybernetics, 2000

Jianfeng Qian, Jianwei Yin, Dongcai Shi, Jinxiang Dong, “Exploring a Semantic Publish/Subscribe Middleware for Event-Based SOA”, In Proceedings of Asia-Pacific Services Computing Conference, 2008.

Joanne Bechta Dugan and Kishore S. Trivedi, “Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems”, In Proceedings of IEEE Transactions on Computers, Vol. 38, No. 6, June 1989.

Kim C.O., Kim S.S., “An efficient real time deadlock-free control algorithm for algorithm for automated manufacturing systems”, Int.J. Prod. Res., vol. 35, 1997

Kshemkalyani, A.D., “Causality between nonatomic poset events in distributed computations”, In Proceedings of the 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1997

Kumar, S., “Challenges for Ubiquitous Computing”, Fifth International Conference on Networking and Services, ICNS 2009.

Leavitt, N., “Complex-Event Processing Poised for Growth”, IEEE Computer Society, 2009

Lobov A., “Formal Validation of Discrete Automation Systems Applying Structural Reasoning and General Unary Hypothesis Automation Methods”. Dissertation. Tampere 2008. Tampere University of Technology. Publication – Tampere University of Technology. Publication 782.

Lobov A., Popescu C. and Lastra J. L. M., “An Algorithm for Siemens STL representation in TNCES”, In Proceedings of IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2006.

Mark R. Blackburn and Robert T. Busser, “Requirements for industrial-strength formal method tools”, In Proceedings of 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, 1998.

Murata T. “Petri nets: Properties, analysis and applications”, In Proceedings of the IEEE, vol. 77, no. 4, pp 541-580, April 1989.

Nancy G. Leveson and Janice L. Stolzy, “Safety Analysis Using Petr Nets”, In Proceedings of IEEE Transactions on Software Engineering, Vol. SE-13, No. 3, March 1987

Popescu C., “An Approach to Incremental Modelling of Web Services Orchestration”. Dissertation. Tampere 2009. Tampere University of Technology. Publication – Tampere University of Technology. Publication 832.

Popescu C., Lobov A., Martinez Lastra J.L., Cavia Soto M. (2008), “A modeling approach to formally represent service orchestration”, International Journal of Computer Aided Engineering and Technology (IJCAET), Vol. 1, Is. 1, pp. 1-30.

Puttonen, J., Lobov, A. and Martinez Lastra, J.L. (2008) ‘An application of BPEL for service orchestration in an industrial environment’, Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, Hamburg, September 2008, pp. 530-537.

R.Valette, J. Cardoso, D. Dubois, “Monitoring manufacturing systems by means of Petri nets with imprecise Markings”, In Proceedings of IEEE Symposium of Intelligent Control, 1989

Rausch M., Hanisch H-M, “Net condition/event systems with multiple condition outputs”, In Proceeding of Symposium on Emerging Technologies and Factory Automation, Paris, France, 1995

Ravier, Dominique, “A Service Oriented Framework Architecture for Intelligent Video Surveillance Systems”, Fifth International Conference on Digital Telecommunications (ICDT), 2010

Richard Zurawski and MengChu Zhou, “Petri Nets and Industrial Applications: A Tutorial”, In Proceedings of IEEE Transactions on Industrial Electronics, Vol. 41, No. 6, December 1994.

Shang Wengli, Duan Bin, Shi Haibo “Event-driven Model for Manufacturing Execution System Platform”,International Symposium on Computer Science and Computational Technology, 2008.

Scherl R., Shafer G., “ A Logic of Action, Causality, and the Temporal Relations of Events”, In Proceedings of Fifth International Workshop on Temporal Representation and Reasoning, 1998.

Sousa P. and Ramos C., “A distributed architecture and negotiation protocol for scheduling in manufacturing systems”, Computers in Industry (1999), pp 103-113

Tang Yongzhong, “Pervasive high reliable monitor and alert system based on EDA”, Joint Conferences on Pervasive Computing (JCPC), 2009

Urban S.D., Biswas I., Dietrich S.W., “Filtering Features for a Composite Event Definition Language”, International Symposium on Applications and the Internet, SAINT 2006.

V.S. Srinivasan and M.A. Jafari, “Monitoring and Fault Detection in Shop Floor using Time Petri Nets”, In Proceedings of IEEE International Conference, 1991

Viswanadham N., Narahari Y., Johnson T.L, “Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models”, IEEE Transactions on Robotics and Automation, 1990

Wysk R.A, Yang N.S, Joshi S., “Detection of deadlocks in flexible manufacturing cells”, In Proceedings of IEEE Transactions on Robotics and Automation, 1991

Y.H. Zhang, Q.Y. Dai, R.Y. Zhong, “An Extensible Event-Driven Manufacturing Management with Complex Event Processing Approach”, International Journal of Control and Automation Vol.2, No.3, September 2009

Yan Liu, Dong Wan, “Complex Event Processing Engine for Large Volume of RFID Data”, In Proceedings of 2010 Second International Workshop on Education Technology and Computer Science (ETCS) 2010

WWW REFERENCES

ESPER, http://esper.codehaus.org

IPC-2541, http://webstds.ipc.org/2541/2541pub.pdf

MOVIDA, http://www.pe.tut.fi/movida3/tools/