

UNIVERSITY OF MASSACHUSETTS DARTMOUTH

Transformation of Live Sequence Charts to Colored Petri Nets (LSCTOCPN)

A Masters Project Report
Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Submitted by
Binsan Khadka

Supervised by
Dr. Boleslaw Mikolajczak
Computer and Information Science Department
University of Massachusetts Dartmouth

January 2007

TABLE OF CONTENTS

ABSTRACT	3
ACKNOWLEDGEMENT	4
MOTIVATION	5
1. INTRODUCTION.....	7
2. SEQUENCE DIAGRAM.....	8
2.1. VARIANTS OF SEQUENCE DIAGRAM	8
2.1.1. UML Sequence Diagram	8
2.1.2. Message Sequence Charts	8
2.1.3. Live Sequence Chart.....	9
2.2. SCOPE OF SEQUENCE DIAGRAM	12
2.3. SEQUENCE DIAGRAM TOOLS	12
2.3.1. Rational Rose	12
2.3.2. Rhapsody	13
2.3.3. Play Engine	13
3. PETRI NETS.....	15
3.1. COLORED PETRI NETS.....	16
3.2. OPERATORS OF PETRI NETS	17
3.2.1. Refinement and Composition.....	17
3.3. APPLICATION OF PETRI NETS	19
3.4. PETRI NETS TOOLS: CPN TOOLS	19
4. RELATIONSHIP BETWEEN INFORMAL APPROACH AND FORMAL APPROACHES.....	20
4.1. FROM UML SEQUENCE DIAGRAMS AND STATE-CHARTS TO ANALYZABLE PETRI NET MODELS	20
4.2. MAPPING UML DIAGRAMS TO A PETRI NET NOTATION FOR SYSTEM SIMULATION	20
4.3. INTEGRATION OF OBJECT ORIENTED DESIGN AND COLORED PETRI NETS USING ABSTRACT NODE APPROACH ..	21
4.4. APPROACHES IN UNIFYING PETRI NETS AND THE OBJECT ORIENTED APPROACH	22
4.5. TURNING HIGH-LEVEL LIVE SEQUENCE CHARTS INTO AUTOMATA	22
4.6. MAPPING LIVE SEQUENCE CHARTS TO COLORED PETRI NETS FOR ANALYSIS AND VERIFICATION OF EMBEDDED SYSTEMS	24
5. LSCTOCPN: LIVE SEQUENCE CHART TO COLORED PETRI NETS.....	26
5.1. CLASSIFICATION OF LIVE SEQUENCE CHARTS	26
5.2. TRANSFORMATION RULES OF LSC CLASSES	30
5.2.1. Single Universal LSC Model	30
5.2.2. Disjoint LSCs.....	31
5.2.3. Non-Disjoint LSCs.....	32
5.2.4. LSCs with Loop	33
5.2.5. LSCs with Inconsistency	36
6. RESULTS	40
7. CONCLUSIONS	44
8. FUTURE PLANS	45
9. APPENDIX.....	46
9.1. LSCTOCPN TOOL.....	46
9.1.1. Package PlayEngine.....	47
9.1.2. Package CPNTools.....	47
9.1.3. Package UMASSD.LSCTOCPN	47
9.2. PLAYENGINE WORKSPACE SCHEMA	47
9.2.1. Workspace (.ews).....	48
9.2.2. Application (Appxyz.xml).....	48
9.2.3. Application Extension File (xyz.ext).....	48
9.2.4. Specification File (Specxyz.xml).....	48
9.3. CPN TOOLS SCHEMA.....	48
10. REFERENCES	50

Abstract

There are various approaches in developing software systems. In most of them, a specification is created before building the software. This approach avoids confusion at the later phase of software development. A simple way of writing the system specification is in the form of words (essay). Software engineer finds it difficult to infer the logics and behavior of the system under development (SUD) because writing gets very vague at times. To overcome this problem, Unified Modeling Language (UML) is used to provide a graphical notation for representing system's behavior. However UML model does not provide a formal approach for documenting specification and has a minimal support for the validation.

There are other ways of formally writing the system specification which can be verified using mathematical techniques and tools. Colored Petri-Nets is one of the formal specification techniques that can be used to model the system's behavior and check its correctness by analyzing liveness, boundedness, and deadlock properties [9]. Colored Petri Nets (CPN) can be modeled and analyzed using CPN Tools.

Thus before developing, software architect needs to create requirement specification in two formats. One format is to communicate with customers (like UML Sequence Diagram) and other format is for analysis and verification purpose (like Colored Petri Nets). An automated tool that can transform from UML like constructs to mathematical model such as Colored Petri-Nets might eliminate the redundancy of writing specification twice.

A Tool LSCTOCPN has been developed as a proof of concept which can be used to reduce the gap between informal method of software specification model and formal methods. This tool reads the Live Sequence Chart (LSC) Model of SUD as an input and transforms the system's behavior into a unified Colored Petri Nets (CPN) model. LSC is a Richer Construct than UML Sequence Diagram. The resulting CPN model of LSCTOCPN tool can be further analyzed using CPN Tools to identify whether the model satisfies the desired properties.

Acknowledgement

I would like to thank my supervisor Dr. Boleslaw Mikolajczak for his continuous support and valuable suggestions for this research work. His insights on the topic of my project were very valuable to me, especially his guidance and feedback to drive me in the proper direction of the research is highly appreciable.

I would also like to thank my friends and colleagues for the great time I had at the Concurrent Software Systems Laboratory. Their moral support at the time of completion of project, presentation and documentation meant a lot to me.

Last but not the least I would like to thank my parents for their love and support.

Motivation

An article written by David Harel in 2001 [1] demonstrates a key point in the field of Software Engineering. His dream to convert *Play-In Scenarios to Code* will be an important contribution for the software development community. For reactive and event driven systems his proposal to convert behaviors into code is the main concern of my project. A Classic system development life cycle and proposed futuristic system development cycle is shown in Figure A and Figure B, respectively.

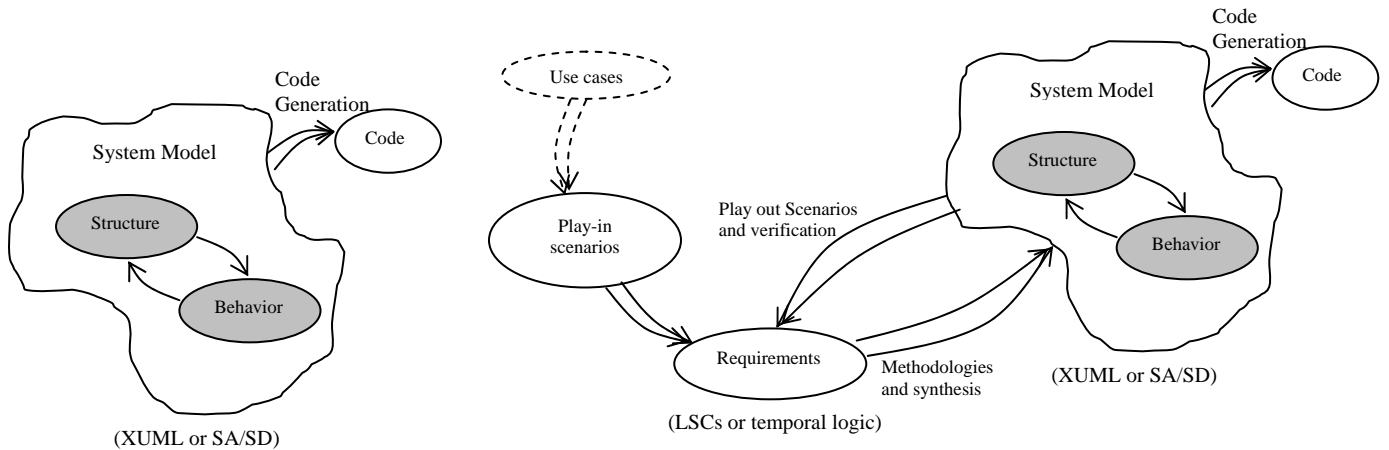


Figure A Classic system development life cycle

Figure B complete system development dream

In Figure A, the system model consists of structure and behavior depicted using visual formalisms in executable UML or structured analysis/structured design (SA/SD). A rigorous semantical basis makes possible to automatically generate fully running code from the model. The dreamt model consists of a convenient way to set up behavioral requirements which is suitable for both system engineers and customers (often customers are non-technical). The play-in scenarios from the use cases will be represented as a requirement of system model; an automated synthesis from these scenarios to system model or code and play-out or verification of system model is all about the dream which is shown in Figure B.

In software development life cycle, conversion from requirements to code is difficult. Often the dependent component is missing or not fully identified in the requirements. This is because the tool used for representing system model is not capable to distinguish between a consistent and inconsistent model. Certainly we need to go through the phase of inconsistency before coming up with a consistent model but what happens if inconsistencies exist in the finalized requirement specification, the answer is: it will be identified at the time of implementation or testing. However the consequence of such identification is expensive as compared to building a system without

any errors in the system model. How nice it would be to build a plug-in for tools used for representing system model before implementation that can verify or explain that the model is correct?

1. Introduction

In software engineering field, there are two approaches for representing the system model. The first and traditional approach is an informal one, which uses a common graphical notation for representation so that it is easier to understand and communicate. The second and important way to specify the software model is by using a formal method, which is built on strong mathematical notations and proofs. This approach is commonly used in mission critical and real time applications like in space exploration, complex medical systems, airplanes and ships where failure of either its part or the entire system will have very bad consequences. One way to avoid failure in this system is by simulating the prototype of this model or the other way is by proving that the algorithms and logics used in this system are correct or both.

Before building a real time system, the system model should be built using both formal as well as informal approach. Informal model is used as a base for development, but the process to build the system is often delayed until the formal model of the system is verified. In the current era of software engineering, formal models are constructed manually from either an informal model or the requirement specification. Due to human intervention in this approach errors are natural and inevitable which might violate the behavior of the system. So there is a gap between these two models.

A Tool LSCTOCPN (Live Sequence Chart to Colored Petri Nets) has been developed as a proof of concept which can be used to reduce the gap between informal method of software specification model and formal method. This tool reads the LSC model of System under Development as an input and transforms the system's behavior into a unified CPN model. LSC is a Richer Construct of UML Sequence Diagram. The resulting CPN model of LSCTOCPN tool can be further analyzed using CPN Tools to identify whether the model satisfies the desired properties.

The organization of this report is as follows: Section 2 and section 3 gives background and discussion about Sequence Diagrams and Petri Nets respectively. Section 4 summarizes the work done by other people. Section 5 explains the method of transformation from LSC to CPN and Section 6 gives the result (verification and validation) of transformed CPN model. Section 7 and 8 concludes the report with suggestions and key points for future work. In Appendix, an explanation of the component of developed tool (LSCTOCPN) is shown followed by description of XML schemas used for the source and destination formats.

2. Sequence Diagram

Sequence Diagram is used to represent the life cycle of an object or the sequence of interactions between objects by message passing. Life cycle of an object is represented by a vertical line and the interaction between objects is represented by a horizontal line with an arrow pointing towards the receiver object. In this section we discuss about common types of Sequence Diagrams: UML Sequence Diagram, Message Sequence Chart and Live Sequence Chart. The shape of arrow, elevation of message communication line has different meaning in different variants of Sequence Diagram. System Model that is designed using the semantics of Live Sequence Chart is an input to the LSCTOCPN tool.

2.1. *Variants of Sequence Diagram*

2.1.1. UML Sequence Diagram

A Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence Diagram. This makes the Sequence Diagram a very useful tool to represent the dynamic behavior of a system. Most of the tutorials and instructions for building UML Sequence Diagram focus on the graphical notation for representing behavior of software. UML Sequence Diagram version 2.0 uses some concepts of Message Sequence Chart defined by International Telecommunication Union [3]. The difference between UML Sequence Diagram and Message Sequence Chart is in their application domain. The former one is primarily intended to be used to model a software system and the latter one for telecommunication systems, however there are very few differences between UML Sequence Diagram version 2.0 and message Sequence Diagram. Sometimes people use Sequence Diagram and Message Sequence Chart interchangeably.

2.1.2. Message Sequence Charts

Message Sequence Charts (MSC) is a graphical language to represent how the messages between instances of system component are exchanged. Each MSC represents a particular scenario of communication between objects. There can be different scenarios among same set of objects based on the nature of communicating messages. MSC can also represent restrictions on transmitted data values and on the timing of events.

The Telecommunication standardization sector of International Telecommunication Union (ITU) has provided a formal definition of the semantics of Message Sequence Charts [3, 4]. This definition provides unambiguous interpretation of Message Sequence Charts.

Expressed in Backus-Naur Form (BNF) grammar, the formal definition of MSC is useful for both developer and user. This document also provides the basic idea to inherit its feature in building a tool that can be used to design behavior using MSC notations. Below is a short extract of a grammar of graphical representation of MSC from page 21 of [3].


<msc diagram> ::=	<simple msc diagram> <hmsc diagram>
<simple msc diagram> ::=	<msc symbol> contains <msc heading> <msc body area>
<hmsc diagram> ::=	<msc symbol> contains { <msc heading> [<containing clause>] } <hmsc area>
<msc symbol> ::=	<frame symbol> is attached to { <def gate area>* } set
<frame symbol> ::=	

Table 1 Syntax of MSC defined by ITU

2.1.3. Live Sequence Chart

As mentioned in [2], even though the ambiguous meaning of Message Sequence Charts (MSCs) has been avoided in a standard recommendation of ITU [3], several fundamental issues have been left unaddressed in the definition of Message Sequence Chart. One of the fundamental issues is: "What does MSC specification mean: does it describe all behaviors of a system, or does it describe a set of sample behaviors of a system?" Other unanswered questions of MSC are listed below that are discussed in [3]:

- ❑ Existential or Universal view: MSC shows only one sample run of the system, one scenario, i.e. it is not possible to specify a mandatory protocol between the communicating entities.
- ❑ Safety and Liveness properties: The semantics of MSCs offers no distinction, whether progress is enforced or not, i.e. The Semantics of MSCs only define permitted sequences of events; the occurrence of an event can not be enforced. MSCs can only express safety (nothing bad ever happens), but not liveness properties (something good will happen eventually).
- ❑ Semantics of conditions: Conditions in MSCs have no formal semantics. As mentioned in the Annex of ITU-T Z.120 Recommendation [4]: "The semantics of a chart containing conditions is simply the semantics of the chart with the conditions deleted from it.". This is obviously not the way to treat conditions from a more formal point of view.
- ❑ Simultaneous events: MSCs do not allow more than one event to happen exactly at the same time, i.e. there is no notation of simultaneity.

- ❑ Activation time: An MSC does not state explicitly when the behavior it describes should be observed, i.e. there is no indication of when the MSC should be activated during a system run.
- ❑ Time treatment: The treatment of time is only rudimentary, since quantitative timing is not covered by the semantics, i.e. timer durations are ignored. Only the correct sequence of timer events, respectively intervals is enforced.

To answer these questions, a new variant of Sequence Diagram termed as Live Sequence Charts (LSC) has been introduced. Details about LSC can be sought from [2][16][24]. A brief overview about the basic LSC features is presented below.

The distinction between mandatory and possible behavior is achieved in LSC by defining a LSC which belongs to either one category or the other. This distinction is achieved graphically by using solid line for mandatory LSC element and dashed lines for possible ones. Mandatory scenarios are classified as Universal LSC and possible one as Existential LSC.

Instances and Messages notations of LSCs have been derived from Message Sequence Chart. Messages of LSCs are of three kinds: asynchronous, synchronous and instantaneous. Synchronous and Instantaneous messages are used interchangeably. Asynchronous messages are visualized by half stick arrows whereas instantaneous messages are visualized by arrows with solid heads. The difference between synchronous and asynchronous message is that synchronous messages (msg1 of Figure 2-1) are drawn horizontally to indicate simultaneity of sending and receiving while asynchronous messages (msg2) are drawn slanted to indicate the passage of time between sending and receipt.

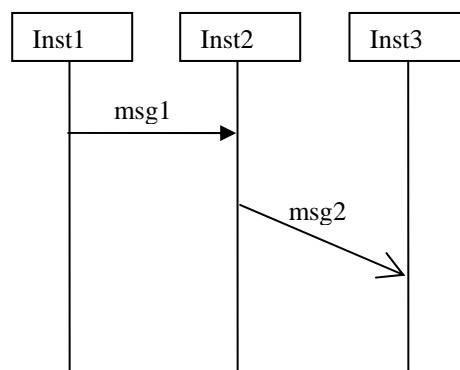


Figure 2-1 A Live Sequence Chart with instances and messages

Liveness and Temperatures Liveness (enforcing the progress) in LSC is shown by associating Temperature with locations and messages. Location of an instance is the place where some event is attached, for example sending or receiving of a message, entering or exiting from the chart body, etc. Every messages and locations are assigned with either a Hot or a Cold

Temperature. A hot location means the progress is enforced, i.e. the chart should eventually progress from a hot location or a hot message has to be delivered. Graphically hot temperatures are drawn by solid lines and cold temperatures by dashed lines. In Figure 2-2 the location (represented as black dot) of instance Inst2 between receiving of msg1 and sending of msg2 (drawn as dashed line) are cold locations.

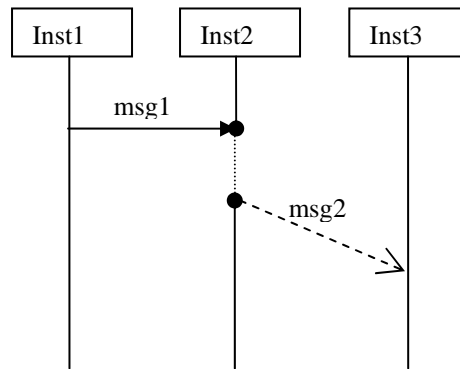


Figure 2-2 A Live Sequence Chart showing temperature & liveness

Negative scenario or anti-scenario is a way of expressing safety (i.e. restricting the unwanted behavior). Anti-scenario can be expressed as a suffix to the Live Sequence Chart by specifying it at the bottom of the Chart. Anti-scenarios are specified in Universal LSCs and mostly associated with a hot temperature. For example once a coin is inserted and coffee is selected in a coffee vending machine, then the system should never return back the money while preparing coffee. This anti-scenario is shown in Figure 2-3.

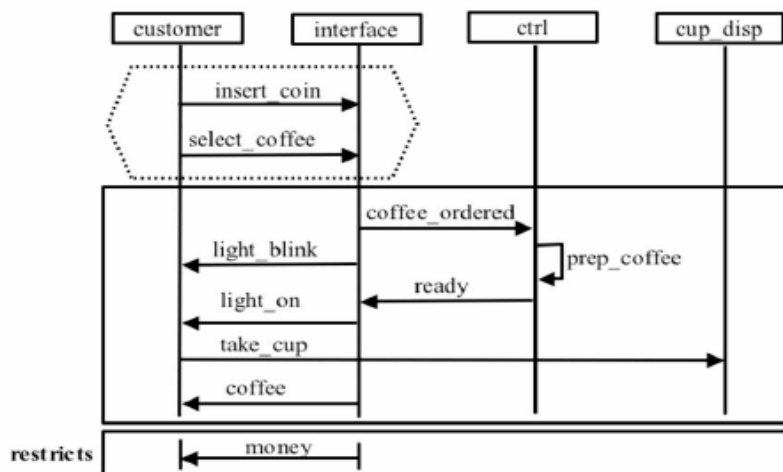


Figure 2-3 A Live Sequence Chart showing anti-scenario

Some other terminologies of LSCs defined to overcome the limitations of MSCs are conditions & local invariants, simultaneous regions & coregions, activation, quantification & pre-charts and time [2].

2.2. Scope of Sequence Diagram

Sequence Diagram is used to model the flow of message communication and it enforces two rules: *ordering of events* and *ordering of messages*. Ordering of events states that events are ordered from top to bottom, i.e. an event in a single instance (horizontal line) and can occur only if all the events above it have already occurred. Ordering of message states that a message must be sent before it is received. Since Sequence Diagrams is a graphical representations, it is easier to comprehend and hence it is used to express the behavior of a system. For example, to know what will happen when you do this ...? What will happen after you do that ...? is easier to show graphically than to explain in words.

2.3. Sequence Diagram Tools

2.3.1. Rational Rose

Rational Rose is a platform that facilitates object-oriented analysis and design allowing system analysts, software developers, testers and project manager to create, view, modify and manipulate elements in Unified Modeling Language. Rational Rose supports use-case-driven object modeling and exposes software development problem in the early stage of development life cycle. Many existing software development groups are using Rational Rose as a tool to design, document and analyze their product.

Rational Rose allows user to design and draw System Specification in Unified Modeling Language (UML) notation. This tool supports drawing of a structural model (such as class diagram, object diagram), behavioral model (like use case diagram, interaction diagram, activity diagram, state chart diagram) and architectural model (collaboration diagram, component diagram, deployment diagram). Rational Rose performs a cross-check between static view (class diagram) and dynamic view (use case diagram and interaction diagram) to find out if the class diagram is not conflicting with interaction diagram or vice versa. Since UML models are not self executable, behavioral analysis can not be easily acquired from UML modeling tool. This tool is only able to check conditions such as: whether a class participates in at least one scenario, and if the operations defined in class are used at least once, etc. The tool does not say anything about the UML model for concurrent software models. It is useful for a system developer to design and develop the system but does not fully help the system analyst of concurrent system. This tool has been widely used in educational as well as in commercial sector.

A Sequence Diagram can be drawn using Rational Rose. Unlike other UML modeling tools such as Class diagram and object diagram, Sequence Diagram has no contribution towards the code generation. The Sequence Diagram in Rational Rose plays a passive role in the system model. Rational Rose does not support execution of Sequence Diagram (since UML does not discuss

execution of Sequence Diagram) which could be useful for checking the correctness of the model or to simulate the system to see whether the result of simulation produces expected behavior.

2.3.2. Rhapsody

Rhapsody is the industry's leading Model-Driven Development environment based on UML 2.0 that allows full application generation for embedded software platform. The code generated from the model drawn using Rhapsody is another view of the model, which allows the developer to make changes at the model or source level and have either dynamically updated. This dynamic model/code associativity gives developer the flexibility to design at any level of granularity, and ensures the model and documentation is consistent with the code. This tool is also widely used by many commercial companies in different fields such as military/aerospace, transportation, telecommunication and medical industries.

This tool not only supports the UML standards, but also some other standards such as OMG sysML, MDA, CORBA, CMMI, AUTOSAR [27]

UML 2.0 has provided some significant steps to capture large-scale architectures. In addition, the Model-Driven Development (MDD) environment of Rhapsody fills the gap between functional methods and UML real time by introducing functional block modeling. According to the product description of ILOGIX [27], they have taken great care to ensure the Rhapsody key enabling technologies of model execution, full production code, etc.

2.3.3. Play Engine

Play Engine is a tool that supports the Live Sequence Chart. This tool allows user to design the Live Sequence Chart Model of a System and also provides a mechanism to execute the modeled system. The term used for these two applications of Play Engine is known as Play-in and Play-out. *Play-in* means to create the Live Sequence Chart for each use case of the system and Play-out means to execute the modeled system and simulate the behavior defined by the Live Sequence Chart. Inconsistencies in the modeled system can be identified by *playing out* the system model.

As opposed to other Sequence Diagram tools, this tool supports a novel approach of documenting the sequence chart by playing with the system's user interface. A skeleton user interface is built on Visual Basic by using two plug-ins of Play Engine: "GUIEdit" and "FuncEdit". GUIEdit is used to create a component of the user interface and FuncEdit to define the function prototype. Once a user interface is ready, it is easier to specify the consequences of events that occur as a result of user's or environment's interaction with the system's component.

The tool Play Engine is a result of research product with successful implementation of Live Sequence Chart Models for some of the commercial and real-time systems. However this product requires improvement before it can be fully utilized by commercial sector. One of the success stories of system modeling using Play Engine and Live Sequence Chart is “Applying LSC to Air Traffic Control Case Study” [7].

3. Petri Nets

Petri Nets was first coined by Carl Adam Petri at early 60's. Here the basic aspects of distributed systems are represented both mathematically and conceptually. Petri Net consists of *places*, *transitions*, and *arcs* [8]. Places are represented as circle, transition as rectangle and arcs as a directional line connecting from place(s) to transition(s) or vice versa but adding an arc between two places or transitions is not the correct way of representation.

A place can contain tokens. A Petri Net model with different number of tokens in each place is a marking which represents particular instance of the designed system (state of a system). Transitions are active elements that are fired when they are enabled (i.e. when precondition associated with the transition is fulfilled). Firing of transition results in decrement of some tokens from all input places of that transition and increment of some tokens at all output places. The number of tokens added or removed from associated place is based on the expression associated with the arc connected from these places to the firing transition.

Formally Petri Nets can be defined as a 4-tuple

$$PN = (P, T, F, M_0)$$

Where,

P is a finite set of places

T is a finite set of Transitions

$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs called flow relation

$M_0: P \rightarrow \{0, 1, 2, \dots\}$ is the initial marking

$$P \cap T = \emptyset \text{ and } P \cup T \neq \emptyset$$

The semantics of Petri Nets can be used to design models of concurrent, distributed, parallel systems and also for the systems showing stochastic property. Petri Nets model of a system can be represented graphically (used as a visual communication for the development team) as well as mathematically (governs the behavior of system and helps in determining the correctness of the model).

Based on the behavior, Petri Nets is classified into different categories. If there is a firing delay associated with each transition and the distribution of such delay is random, the resulting net belongs to class of Stochastic Petri Nets.

Petri Nets being a Graphical means of representation, the models for larger systems might result into a big picture not easy to trace by individual and difficult to understand. However Hierarchical

Petri Nets can simplify the problem of understandability in which a transition or place at top level is sub-divided into detailed Petri Nets that is represented in separate page. There is another class of Petri Nets which compresses the representation, called as Colored Petri Nets (CPN). In CPN a color is attached to each token and a multi set of colors in each place. More about Colored Petri Nets is discussed in next section.

3.1. Colored Petri Nets

Colored Petri Nets (CPN) is a high level Petri Nets which describes complex systems in a manageable way. In this model each token is attached with a data type called *color*. A color might be as simple as an *integer* type representing the number of tokens. A complex type can be collection of simpler data types and/or complex data types.

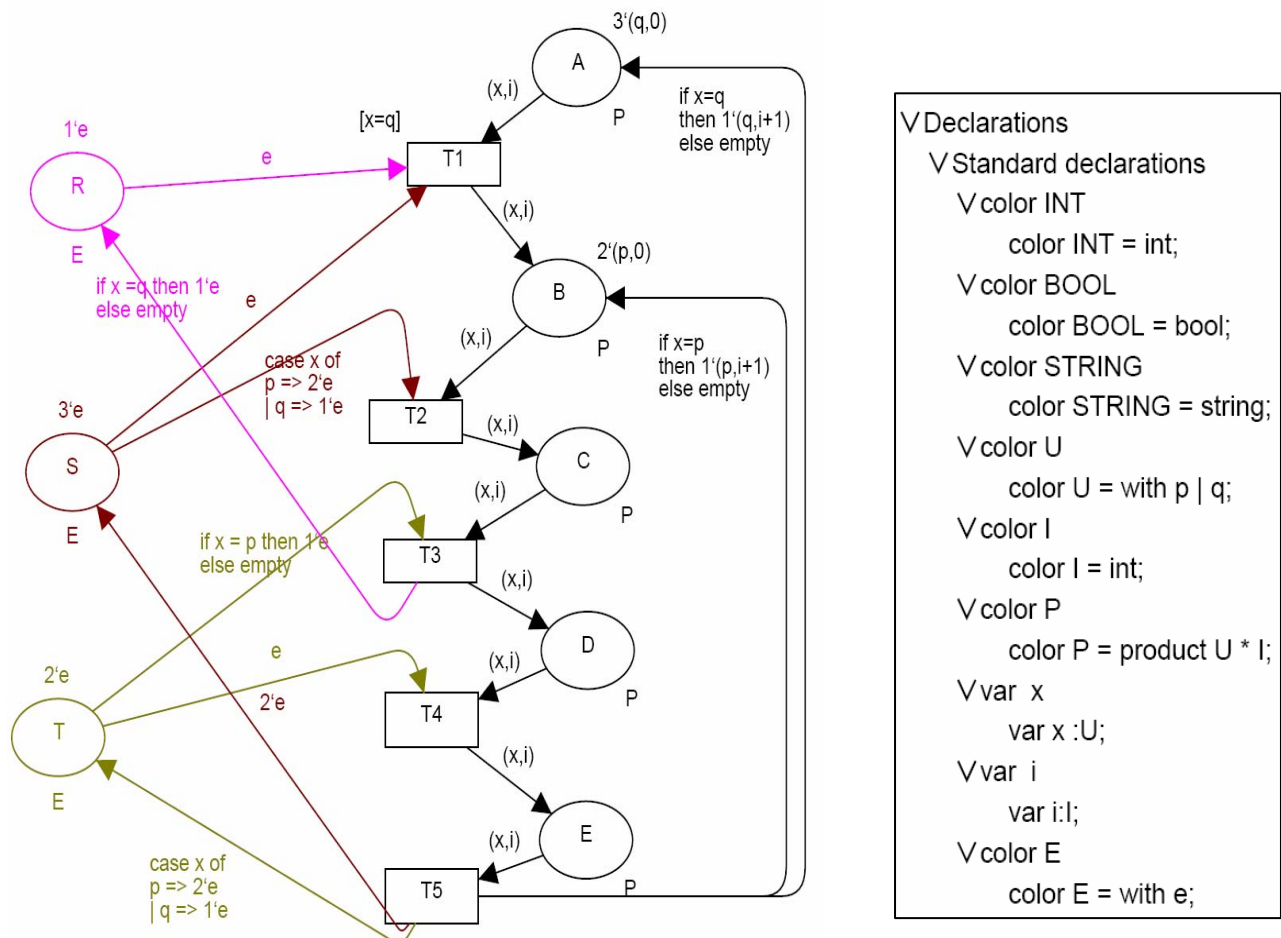


Figure 3-1 CPN model of resource allocation system; variable declarations on the right

Figure 3-1 is a Colored Petri Nets model of resource allocation system described in [25] which was drawn in CPN Tools. The box on the right side contains the declaration of color of the given Petri Nets model.

Initial state of CPN model is said to be an initial marking. After a single step (when the transition occurs), there will be a change in instance associated with some of the places and the resulting Petri Net represents a new state. A series of markings in sequence starting from initial marking to the end is termed as occurrence sequence.

A Petri Nets design can be analyzed to find possible flaws in the early stage by executing the model a number of times with different values of variables. According to Jensen [9], Colored Petri Nets possesses dynamic properties that characterize the behavior of individual CP-nets. Some of them are Boundedness property, Home Property, Liveness Property and Fairness Property [9].

3.2. Operators of Petri Nets

3.2.1. Refinement and Composition

Petri Nets support a mechanism to build a hierarchical Petri Nets model using abstraction or refinement technique. We use the concept of bordered sets to explain abstraction of Petri Nets. There are two types of bordered sets: place bordered set and transition bordered set.

Let $N = \{P, T, F\}$ be a net, $Y \subseteq \{P \cup T\}$ be a set of elements. Then $\partial(Y) = \{y \in Y \mid \exists x \notin Y. x \in \text{loc}(y)\}$ is the bordered set of Y . Where, $\text{loc}(y)$ is locality defined as a set of elements which includes y and all the places or transitions immediately connected with y . Y is called place-bordered if $\partial(Y) \subseteq P$ and transition bordered if $\partial(Y) \subseteq T$. Figure 3-2 is an example of transition bordered set where all the arcs crossing the boundary of set are connected to transitions of the bordered set. The bordered set is represented by the Petri Nets structure inside the dashed region.

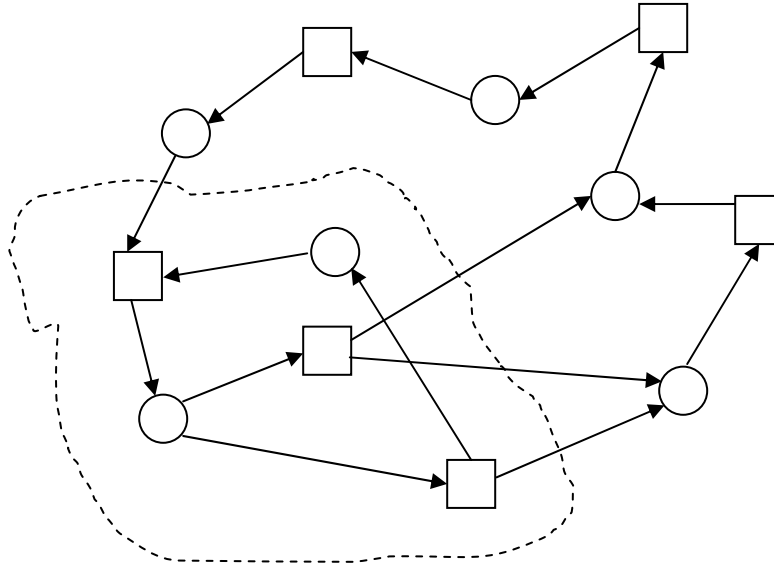


Figure 3-2 an example of transition bordered set

This Transition Bordered Set can be refined by representing the transition bordered set with a single transition. The result of abstraction is again a net structure. Similarly a Place Bordered set can be refined by replacing it with a single place.

Composition of Petri Nets can be achieved by fusion of places or transitions, termed as place fusion and/or transition fusion. For example the first three figures shown on the left side of Figure 3-3 are the different scenarios of fusion of places and the first three nets shown on the right side of Figure 3-3 are the scenarios of transition fusion. The fourth net is the resulting net achieved after performing fusion on any of these three cases.

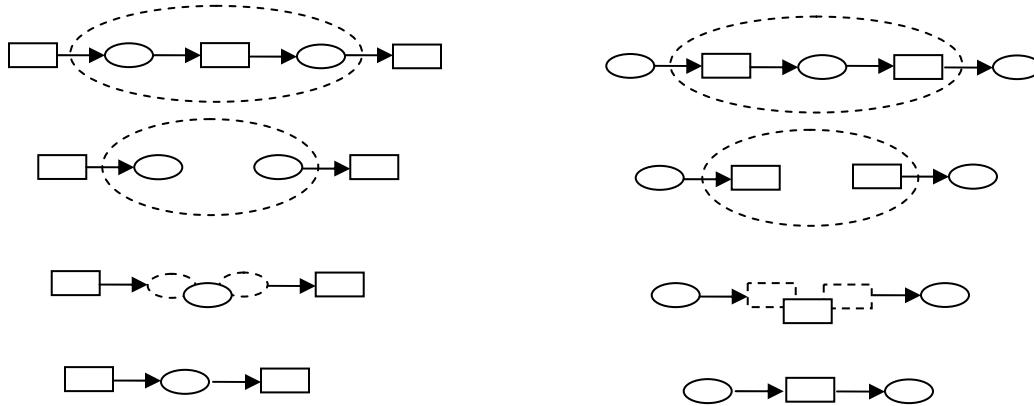


Figure 3-3 Scenario of Place and Transition fusion

Folding is a special case of abstraction, known as strict abstraction. A strict abstraction is defined as an abstraction if every Y_i is either a set of places or a set of transitions [$Y_i \in Y \subseteq \{P \cup T\}$].

3.3. Application of Petri Nets

Petri Nets have been widely used in different application domains. Some of the common domains include manufacturing, telecommunication and workflow management. Other application domains where Petri Nets have been turned out to be a useful design/analysis tool are distributed software systems, multiprocessor systems, embedded systems, hardware/software architectures, and many other software engineering fields.

UML + CPN @ Nokia [41] is one of the research project performed at the Software Architecture Group (SAG) in the software technology laboratory at Nokia research center. The objective of this project is to combine the static UML models and dynamic CPN model to form an integrated set of models which can be used to investigate the statistics about memory usage by simulating the scenarios using CPN Tools.

3.4. Petri Nets Tools: CPN Tools

CPN Tools is a CASE Tool for editing, simulating and analyzing Colored Petri Nets. The tool features incremental syntax checking and code generation that take place while a net is being constructed. A fast simulator efficiently handles both untimed and timed nets. It uses the CPN ML language for declarations and net inscriptions [29].

Simulation in CPN Tools can be performed by using simulation tools [23] (play, rewind, fast-forward). A Play tool causes one enabled transition to occur, rewind tool is used to return to initial marking, and fast-forward to skip number of steps. This way we can simulate the Colored Petri Nets Model to check the behavior of model in different states (markings).

CPN Tools also facilitate generating and analyzing full and partial state spaces for Colored Petri Nets. The state space tools are EnterStateSpace, CalcSS, CalcSCC. EnterStateSpace tool is used to generate net-specific code necessary for generating a state space, then CalcSS tool is used to generate the state space and CalcSCC tool calculates the strongly connected component graph of the state space.

There are two additional tools SStoSim and SimtoSS that allow switching between simulator and state space. SStoSim takes a user specified state from the state space and moves it to the simulator. Similarly SimtoSS will move the current state of the CP-net in the simulator to the state space. SaveReport tool is used to generate state space report which contains information about one or more of the following: statistics about the generation of the state space, boundedness properties, home properties, liveness properties and fairness properties.

4. Relationship between Informal Approach and Formal Approaches

Some of the related work in finding the relationship between informal approach and formal approach are briefly discussed in this section.

4.1. From UML Sequence Diagrams and State-charts to Analyzable Petri Net Models

Paper [10] proposes an approach to transform UML Statecharts and Sequence Diagram to Labeled Generalized Stochastic Petri Nets (LGSPN) model. This paper demonstrates a generalized approach of translation using abstract syntax of the UML collaboration diagram and state machine package. Author has demonstrated a method to translate the two meta-models of UML, known as statecharts and Sequence Diagrams into Petri Nets model taking into account Synchronous and Asynchronous communication. The target model of this approach, LGSPN model is a variant of Petri Nets that assigns label in places and transitions of Generalized Stochastic Petri Nets [22]. The approach for translation is summarized below:

Each message in a Sequence Diagram is transformed into a labeled generalized stochastic Petri Nets. The resulting net that represents sub-system of destination Petri Nets is composed into a single system, taking into account the partial order relation that exists between messages belonging to same interaction. The translation of individual message into Petri Nets is done separately for both the cases, i.e. when communication is synchronous and when it is asynchronous. The translated Petri Nets can now be used to analyze the behavior of the system by simulating it.

4.2. Mapping UML Diagrams to a Petri Net Notation for System Simulation

Paper [11] discusses a prototype tool mapping systems specified using UML diagrams to Colored Petri Nets notation. The author considers a subset of statechart to generalize the translation of UML specification into net model and further simulates the result of the transformation. Figure 4-1 shows the architecture of approach discussed in this paper.

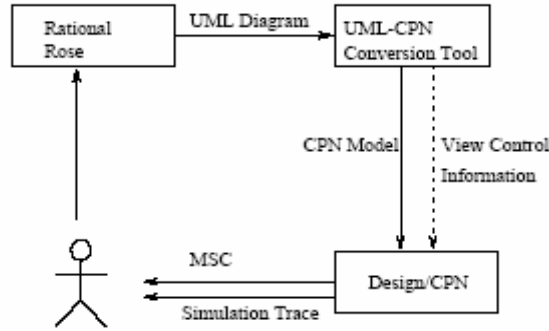


Figure 4-1 Architecture of the approach used by Hu & Shartz

The transformation is discussed by showing the process of transforming Object Net Model (ONM) into a target model structure. An Object Net Model, a derived model from UML state chart describes the behavior of an individual object and defines the token routing mechanism within an object as shown in Figure 4-2. Target Model (Figure 4-3) consists of modules (or pages) with four different types namely object page, INL Page, Main Page and Init Page. Brief introduction of these models are discussed in Section 3.1 and 4.1 of [11].

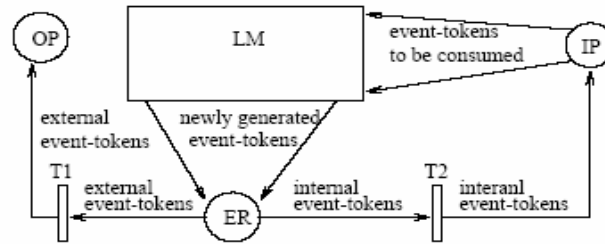


Figure 4-2 Structure of object net model

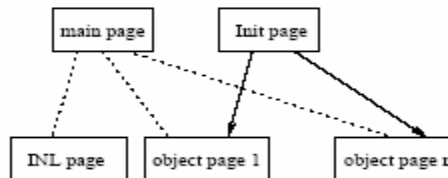


Figure 4-3 Structure of target model

From the statechart model of any system the tool discussed in the paper tries to map each ONM model into an object page; however, the formal method of mapping is not shown by the author. The prototype tool will then try to run simulation and produces Message Sequence Chart for the two events of interest. The case where more than two events are communicating during simulation is also not discussed in this paper.

4.3. Integration of Object Oriented Design and Colored Petri Nets using Abstract Node Approach

Bauskar [14] has discussed about the term “Inheritance Anomaly” for concurrent systems using object oriented design in his master’s thesis. In concurrent systems, inheritance is lost due to disagreement in inheritance specification and the synchronization constraints of concurrent

operations, which is known as Inheritance Anomaly. This requires redefinition of inherited methods in order to maintain the integrity among concurrent objects. Author has proposed a method of pre-conditions and post actions to solve the problem of Inheritance anomaly. Adding a Synchronization constraint in the concurrent object requires the specification of pre-conditions and post-actions which results in avoidance of Inheritance Anomaly.

The approach of removing inheritance anomaly is done for three different cases: State Partitioning, History Only sensitive and State Modification. Further in this thesis, author has chosen model of ATM system to design, verify and validate. The UML model was considered from [12] and for the same system, Colored Petri Nets version of the same system is described.

4.4. Approaches in Unifying Petri Nets and the Object Oriented Approach

In the process of integrating both the object oriented model and Petri Net model, the community from both the approaches tend to use either object inside Petri Net or Petri Net inside objects. Bastide [13] tries to unify both of these approaches into single co-operative objects.

Communications between co-operative objects are made like a Client-Server model. The approach of unification is made starting from <<Petri Net inside object>> and the message passing among objects is explained in the form of request-response pair. When a Transition inside client's object is a method call of some other server then such transition will trigger the method of server object (request) with a wait on client side until server returns the result (response).

4.5. Turning High-Level Live Sequence Charts into Automata

Live Sequence Chart borrows some of the features of Message Sequence Charts to overcome the limitations of MSCs. The limitations of MSCs identified by the author of this paper [6] are: lack of message abstraction, an expressive sub-language for conditions and the ability to express whether a scenario describes an example or a universal rule. To add formal semantics to this language, authors of this paper have composed scenarios in LSC languages and represented it in ω -regular language using which the representation of automata over infinite traces becomes easy. However, in order to achieve the desired results they have defined additional semantics for LSC and hence their input model is a variant of the original Live Sequence chart defined by Damm and Harel [16].

A Chart is categorized into two types: Basic Chart (BCs) and Iterative Chart (ICs). They have added a constraint of cold temperature at the end of each chart to attain successful termination of chart. An invariant has been added to Semantics of Basic Chart in order to specify that a

condition between two locations continuously holds. An invariant is also associated with a temperature. A hot invariant should always hold whereas a cold invariant may be violated resulting in premature, but successful termination of chart. Using these terminologies, a formal definition of Basic Chart is defined using 10-tuple (Definition 2.3 of [6]). One can obtain a Finite Automaton also from the Basic Chart which is defined as 5-tuple (Definition 2.10 of [6]).

Figure 4-4 shows how the basic chart automaton articulates six different types of transition relations [6]. Π_c -stuttering is a step in which chart does not progress, because only irrelevant events occur.

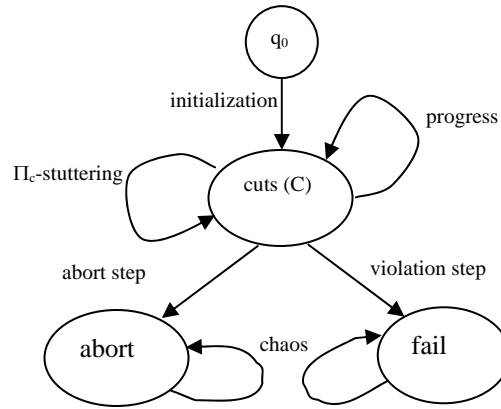


Figure 4-4 How the 6 steps of basic chart's automaton articulate

At run time, in order to remember which locations have already been reached all the visited locations are put into a set known as cut. Thus a cut is a subset of Loc which uniquely represents the current state of a system, where Loc is a set of all possible locations of Live Sequence Charts.

Once the basic chart is defined, iterative charts can be built. Iterative Charts (ICs) provides means to combine basic scenarios (expressed as Basic Charts) to produce more complex scenarios. The constructions are:

- ☐ Conditional branching
- ☐ Sequential composition
- ☐ Parallel composition
- ☐ Nondeterministic choice
- ☐ Iteration

Let us revisit the parallel composition of Iterative charts defined in [6]

Let A_{IC1} and A_{IC2} be two automata of a Live Sequence Chart:

$$\begin{aligned}
 A_{IC1} &= \langle L^1, \{q_0^1\}, T^1, F^1 \rangle \\
 A_{IC2} &= \langle L^2, \{q_0^2\}, T^2, F^2 \rangle
 \end{aligned}$$

$$\begin{aligned}\text{Restricts } A_{IC1} &= \{\Sigma_1 \mid \Sigma_1 \subseteq \Sigma\} \\ \text{Restricts } A_{IC2} &= \{\Sigma_2 \mid \Sigma_2 \subseteq \Sigma\}\end{aligned}$$

Where,

Σ = set of input alphabets (events)
 L^1, L^2 are set of states {each state is associated with valuation s which is an evaluation of observable propositional formula over the domain of truth values.}

$\{q_0^1\}$ and $\{q_0^2\}$ are initial states
 T^1 and T^2 are Transition relations
 $T^1 \subseteq L^1 \times (\Sigma \cup \{\varepsilon\}) \times L^1$
 $T^2 \subseteq L^2 \times (\Sigma \cup \{\varepsilon\}) \times L^2$

F^1 and F^2 are final states of Automata A_{IC1} and A_{IC2} respectively.

& restricts set contains the set of events (or input alphabet) that is recognized by automaton.

The Parallel Composition of two Charts is computed as follows

$$\begin{aligned}A_{IC1||IC2} &= \langle L, \{q_0\}, T, F \rangle \\ \text{Restricts } A_{IC1||IC2} &= \{\Sigma_1 \cup \Sigma_2\} \\ L &= L^1 \times L^2 \\ \{q_0\} &= \{(q_0^1, q_0^2)\} \\ F &= \{(q_1, q_2) \mid q_1 \in F_1 \wedge q_2 \in F_2\}\end{aligned}$$

And,

$$\begin{aligned}[(q_1, q_2), (s, e), (q_1', q_2')] &\in T \\ \text{iff } \{[q_1, (s, e), q_1'] \in T^1 \wedge [q_2, (s, e), q_2'] \in T^2\}\end{aligned}$$

Which means that there will be a transition from state (q_1, q_2) with valuation s to a new state (q_1', q_2') on an input event e if and only if there exist transition between q_1 and q_1' in the first and between q_2 and q_2' in the second chart automata for the same input event e and for the same valuation s in q_1 and q_2 .

4.6. Mapping Live Sequence Charts to Colored Petri Nets for analysis and verification of embedded systems

Paper [15] is pretty much similar to our work, tries to convert the Live Sequence Chart to colored Petri Nets model as an approach for analysis and verification of embedded systems. This approach uses a single object instance of LSC element at a time to obtain a CPN model and then the composition of CPN model is performed by the fusion of common transition names. The composition rule is not clearly discussed in their paper. The result of analysis from the transformed CPN model is discussed for the case study of a pulse-oximeter has been discussed for the different properties of CPN Tools, but it is unclear whether the analysis has been done manually (and possibly error prone) or by using any tools.

Figure 4-5 shows the block diagram of LSC to CPN transformation [15] and additional steps used to check the equivalency of these two representations.

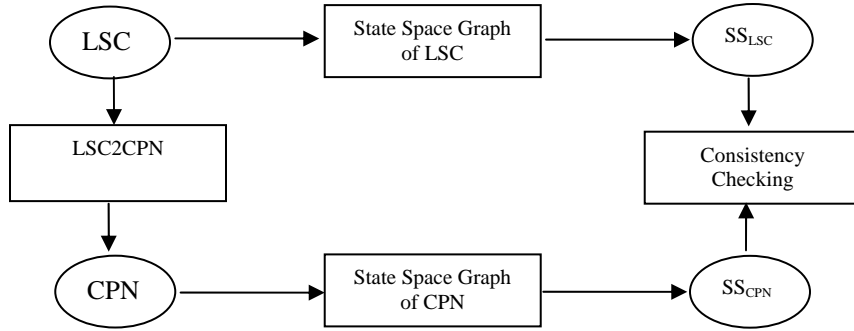


Figure 4-5 Validation Phases of LSC2CPN

The phase “State Space Graph of CPN” is well defined and seems obvious, but the phase “State Space Graph of LSC” and consistency checking are not clearly described in this paper. Without any proof or explanation, the comparison of Petri Nets and LSC semantics in this paper says that Reachability graph of CPN and the Reachability graph of LSCs scenario are bisimilar (branching time equivalence).

5. LSCTOCPN: Live Sequence Chart to Colored Petri Nets

The basic idea of the mapping rule is to create an equivalent Colored Petri Net Models for each Universal Live Sequence Chart model and to use various rules for the classification of Live Sequence Charts in order to compose a single representation model of the system. Composition rules for the related objects of different charts are applied to get a single Colored Petri Net representation of the system. It is not necessary that the single representation model is a single Colored Petri Nets Graph, but all the related objects are interconnected with each other. There is no clear picture of the composition rule for Colored Petri Nets. The composition rule for the Petri Nets of Live Sequence Charts identified as part of this project will be discussed in Section 5.2.

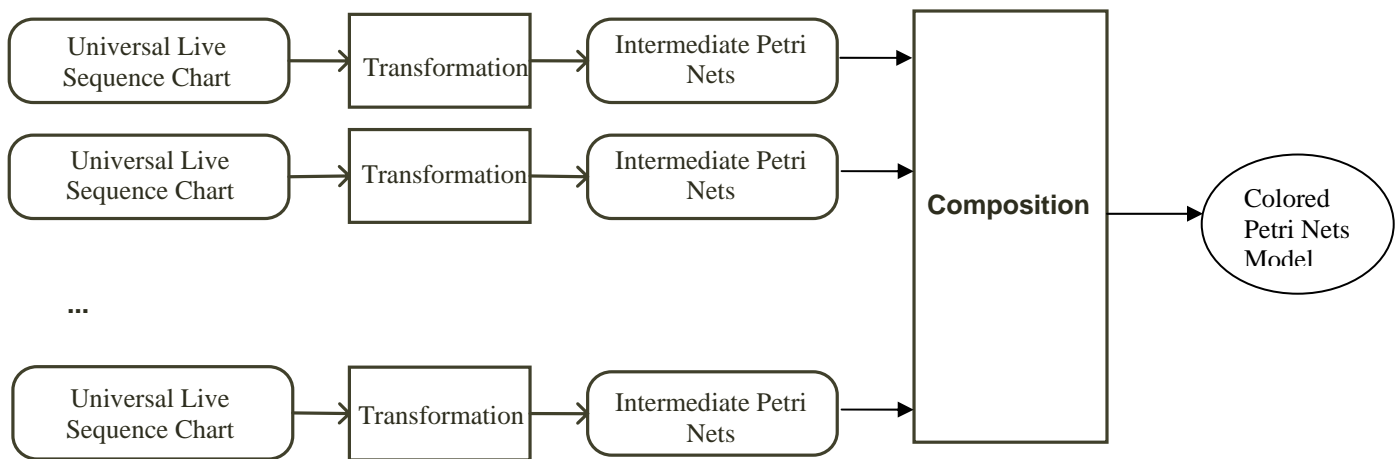


Figure 5-1 Block diagram of transformation from LSC to CPN

5.1. Classification of Live Sequence Charts

After analyzing the properties of Live Sequence Chart, some of the basic classification of Live Sequence Charts has been made and the rules for their transformation are discussed in Section 5.2. First three cases are the distinct classification whereas the last two cases are the special case of Non-Disjoint LSCs.

The five basic classifications are:

- **Single Universal LSC Model:** A single Universal Live Sequence Chart with set of messages will be the basic classification of the transformation model. This model has a Live Sequence Chart with any number of objects communicating with each other via messages. According to the rule of Live Sequence Chart the messages shown in Pre-Chart body of the Live

Sequence Chart are to be successfully executed in order to execute the messages in the Chart Body.

The Live Sequence Chart shown in Figure 5-2 is an example of Single Universal LSC Model, where A, B and C are the objects involved in this chart and message1 and message2 is the communication between these objects in this chart. Message1 is in Pre-Chart body which means that B cannot send message2 to C until it receives message1 from A.

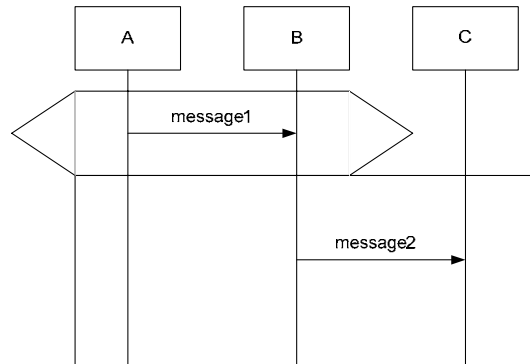


Figure 5-2 Example of a single universal LSC

- ❑ **Disjoint LSCs:** Two LSCs with disjoint objects and messages has no relation with each other. For example in the Figure 5-3 there are two Universal Live Sequence Charts with its own set of objects and messages. In first chart, objects A, B and C are communicating with each other via message1 and message2 and second chart consists of three objects D, E and F and their communicating messages: message3 and message4. Since none of the objects are common in both the charts, these two charts are categorized as disjoint chart.

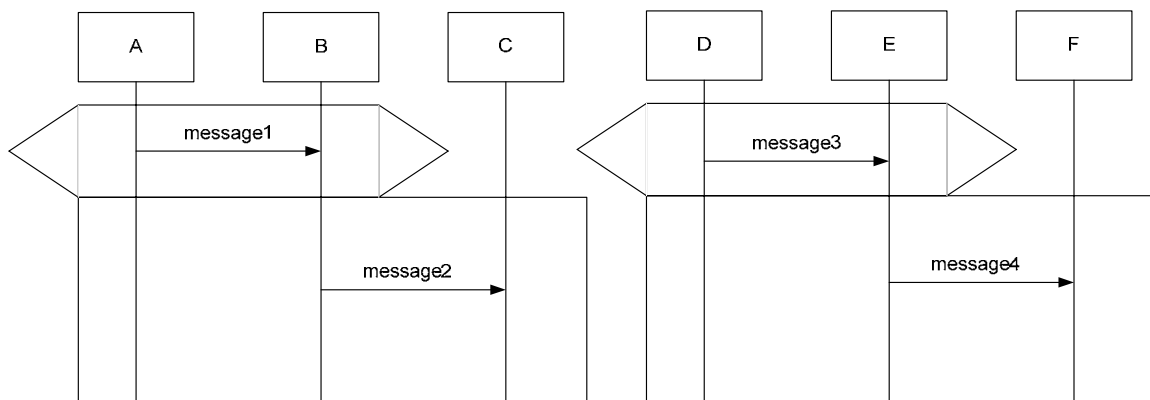


Figure 5-3 Example of disjoint LSCs

- ❑ **Non-Disjoint LSCs** If the participating objects of a Live Sequence Chart are common in two LSCs then they are categorized as Non-Disjoint LSCs. As a basic example, Figure 5-4 has two Live Sequence Charts that will be activated by the same message: message1 from A to

B. It is possible to create different scenarios with some common objects in Live Sequence Charts. Hence such kind of non-disjoint LSCs can be commonly found in the system model.

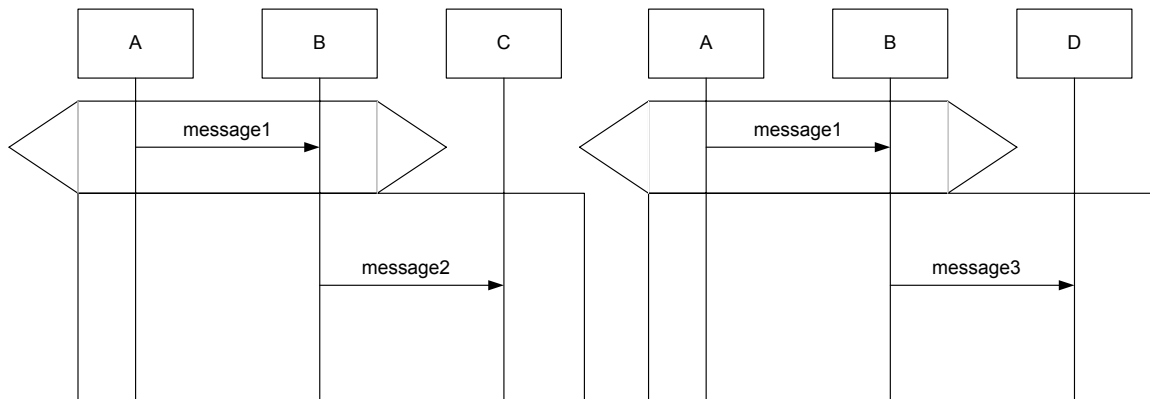


Figure 5-4 Example of Non-Disjoint LSCs

- ❑ **LSCs with Loop** This is a special case of Non-disjoint LSCs showing three Universal LSCs. The behavior of this system model is periodic in nature, which means once the system starts it continuously goes into an infinite loop. Such kind of behavior can be easily seen in the real time system such as clock (which needs to be continuously running once it is started).

To analyze the result of transformation, we have classified this model as a separate entity. In this example, first LSC (LSC1) consists of three entities: User, Antenna and Display and the second (LSC2) and third (LSC3) consist of a single object display which is sending message to itself to change its state ("Show Reception(2)" means change the display of reception to level 2). Once user opens the antenna, it activates LSC1 and thus the display will send a message "Show Reception (2)" to itself which will activate LSC2 and resulting in sending another message: "Show Reception (4)" will now activate LSC3 and hence LSC2 and LSC3 will activate each other after one chart is completed successfully.

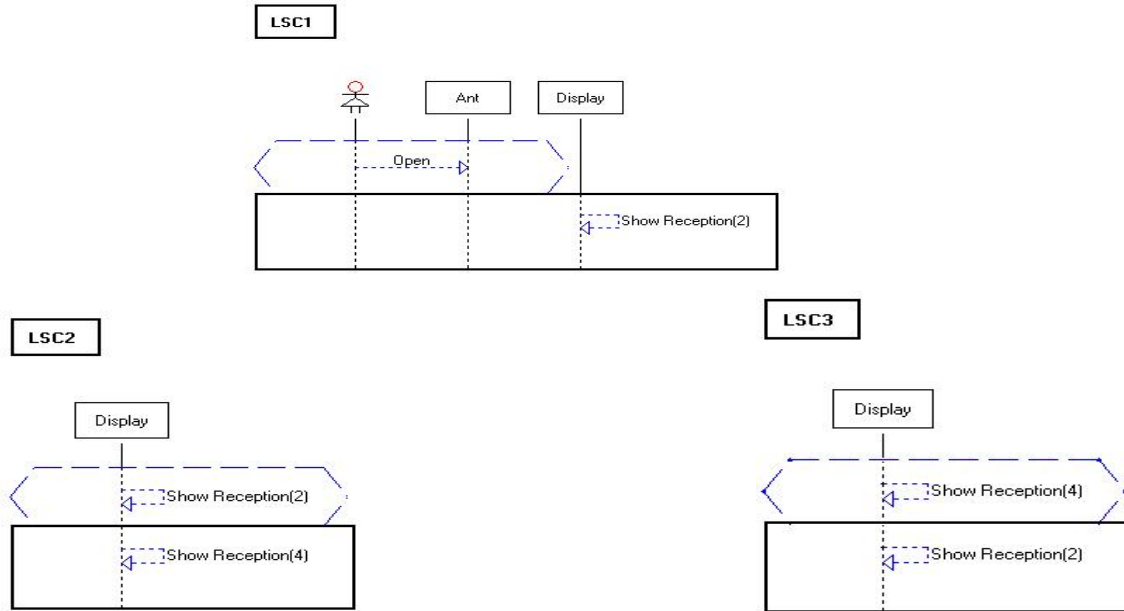


Figure 5-5 Example of LSCs with a loop

- ❑ **LSCs with Inconsistency** This is another special case of Non-Disjoint LSCs with four Live Sequence Charts. An inconsistency in Live Sequence Charts implies that there exists a scenario which contradicts with some other scenario of the same modeled system. If such inconsistency is eliminated, we get a model of a deterministic system where only one system exists at a time. For example a bulb can be either in "ON" state or in "OFF" state. But what if one scenario states that a bulb should be in "ON" state and at the same time another scenario states that the bulb should be in "OFF" state. Such kind of inconsistency says that there is some problem in the system model; either of the two scenarios should be modified to make the system consistent. Figure 5-6 is an example with an inconsistency.

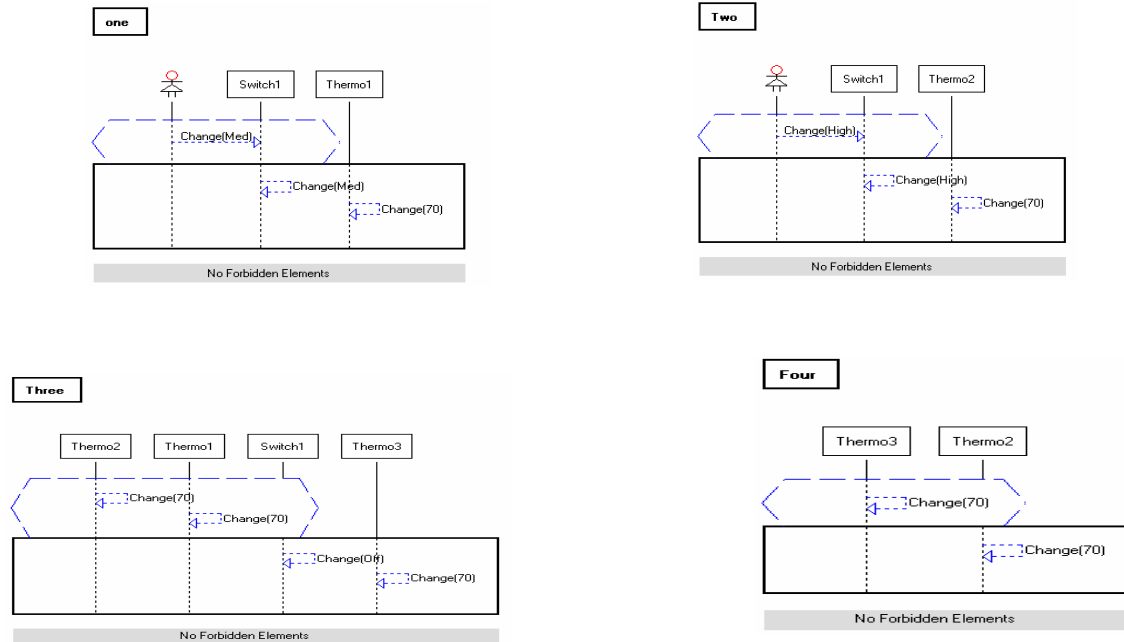


Figure 5-6 Example of LSCs with inconsistencies

5.2. Transformation Rules of LSC Classes

From the five different classes shown above, we will present a rule to transform each of these Live Sequence Chart model into its corresponding Colored Petri Nets. For simplicity the colors of all the places has been defined as of type Integer and different integral values are used to represent a single scenario, i.e. each Live Sequence Chart is a unique scenario and hence every transformed model will have the same number of unique integral tokens. For example in LSCs with loop model there will be three scenarios and hence in Colored Petri Nets model there should be three distinct tokens “1”, “2” and “3” (we will start the token numbers from 1). The explanation of each transformation rules and the corresponding Colored Petri Nets Model are shown below. Each individual Universal LSC model will be first converted to an intermediate representation as outlined in Figure 5-1 and then these intermediate representations are composed into a CPN model. The transformation rules for the five basic classifications are show below:

5.2.1. Single Universal LSC Model

The First rule of Transformation from Live Sequence Charts to Colored Petri Nets is by converting all the messages into transitions and assigning a place before and after each message. These transitions are ordered according to its original ordering in the Live Sequence Chart.

Hence the rule of Single “Universal LSC model is summarized as:

“A Message of Live Sequence Chart is mapped to a transition of a Colored Petri Nets”

The Petri Nets model shown in Figure 5-7 is the transformed Petri Net model of the Live Sequence Chart shown in Figure 5-2. Here the place P1 represents a cut (discussed in Section 4.5) of Live Sequence Chart which includes the locations before sending message1 and the place P2 represents a cut with locations after message1 is received by B and before sending message2. For example the cut of P2 is shown as a horizontal dotted line in Figure 5-8.

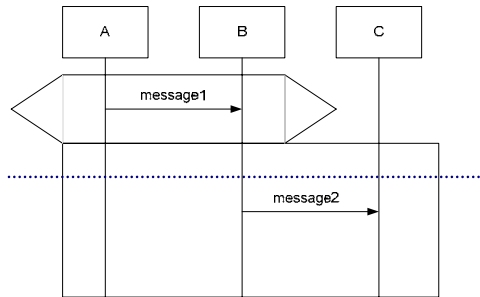


Figure 5-7 Cut in a LSC

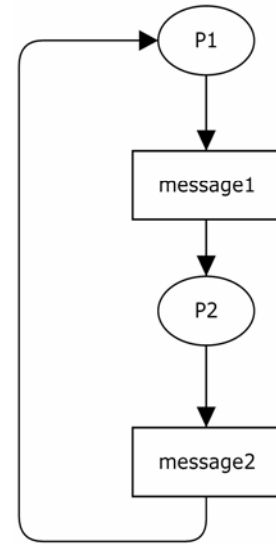


Figure 5-8 Transformed CPN Model of Single Universal LSC

The place P1 in this case represents two cuts of the chart, one is before sending message1 and the other is after message2 is received by C. Both cuts represent that the chart is ready to accept a message: message1 to be sent from A to B, the first one being the initial execution of the chart whereas the second being the execution of the same chart after successfully executing the previous instance of the same chart. A token can be added in place P1 representing that the Petri Nets model of its source Live Sequence Chart is active and the system is ready to fire a transition named “message1” which is equivalent to saying that in Live Sequence Chart, object A is ready to send message1 to object B.

5.2.2. Disjoint LSCs

Using the Rule of Single Universal Live Sequence Chart defined earlier, each LSC will be transformed to its equivalent Colored Petri Nets Model. The transformed model of Live Sequence Chart shown in Figure 5-3 is shown in Figure 5-9. None of the objects and messages is common in both of the Live Sequence Charts; the final CPN model will have two nets which are not connected to each other.

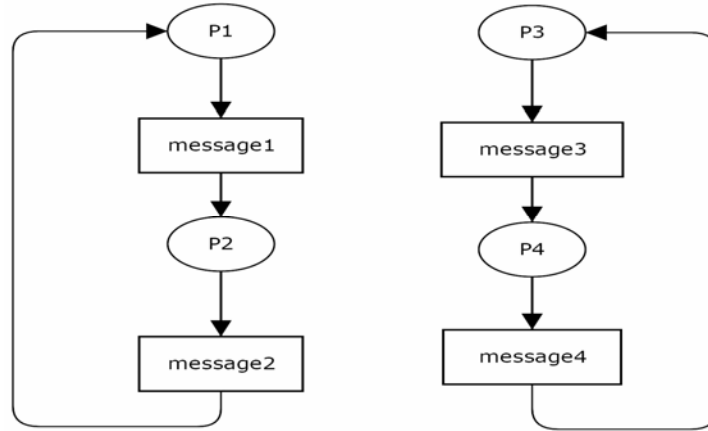


Figure 5-9 Transformed CPN Model of Disjoint LSCs

The rule of Disjoint LSCs is summarized as:

“Two LSCs with no common objects and messages should have no interference with each other.”

5.2.3. Non-Disjoint LSCs

This type of Live Sequence Chart pair where some objects and the message between these objects are common are first individually transformed into intermediate nets which is then later composed into single model. The process of extracting intermediate net from the Non-Disjoint LSC is same as that of Disjoint LSCs assuming that the nets are disjoint. Now these intermediate nets are composed into a single net by merging the transition corresponding to the common messages of Live Sequence Chart. The Intermediate net of LSC of Figure 5-4 is shown in Figure 5-11.

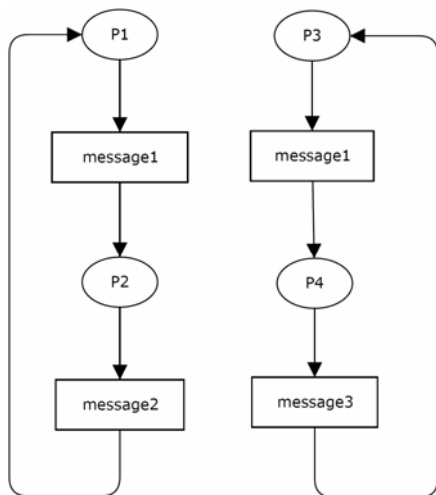


Figure 5-10 Intermediate CPN model of non-disjoint LSCs

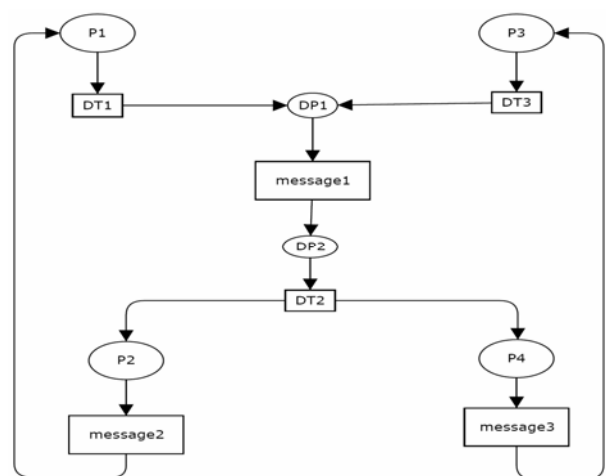


Figure 5-11 Transformed CPN model of non-disjoint LSCs

In the intermediate net (Figure 5-11), two transitions having label *message1* connected from places P1 and P3 are the same message instances of two Live Sequence Charts of Figure 5-4 sent from object A to object B. Hence these two message instances should be composed into a single transition. This composition is achieved by adding a pair of Transition and Message (DT1, DP1) before the common transition and a place and transition pair (DP2, DT2) after the common message, where DP and DT are the dummy places and dummy transitions used to add arcs from and to the places that precedes and follows the common transition as shown in Figure 5-10. The composed single Petri Net model is equivalent to the behavior specified by the Live Sequence Chart of Figure 5-4. The receipt of message1 by object B will make it ready to send message2 to object C and message3 to object D simultaneously which is also the case in the composed Petri Nets model Figure 5-10. At first both P1 & P3 has a token which activates the dummy transition connected to them (DT1 and DT3) which in turn put two tokens at place DP1 and hence activating the transition “message1”. All the dummy transitions are fired as soon as it is activated and puts the token to its connecting places. Once message1 transition is activated, it will automatically activate the subsequent Dummy transition DT2 and eventually activation of message2 and message3 in parallel. The sequence of execution of messages in composed Petri Nets model is the same as that of the LSC model of Figure 5-4.

The transformation rule for Non-Disjoint LSCs is summarized as:

“If two charts are activated by the same message, then it should enable the transition corresponding to the next active message of both charts”

5.2.4. LSCs with Loop

For more than Two LSCs the step to generate intermediate Net is same as case 2 and case 3 where each individual LSC element is transformed into its corresponding CPN Model. The Petri Nets models shown in Figure 5-12 are the intermediate nets of an example shown in LSCs with loop classification Figure 5-5.

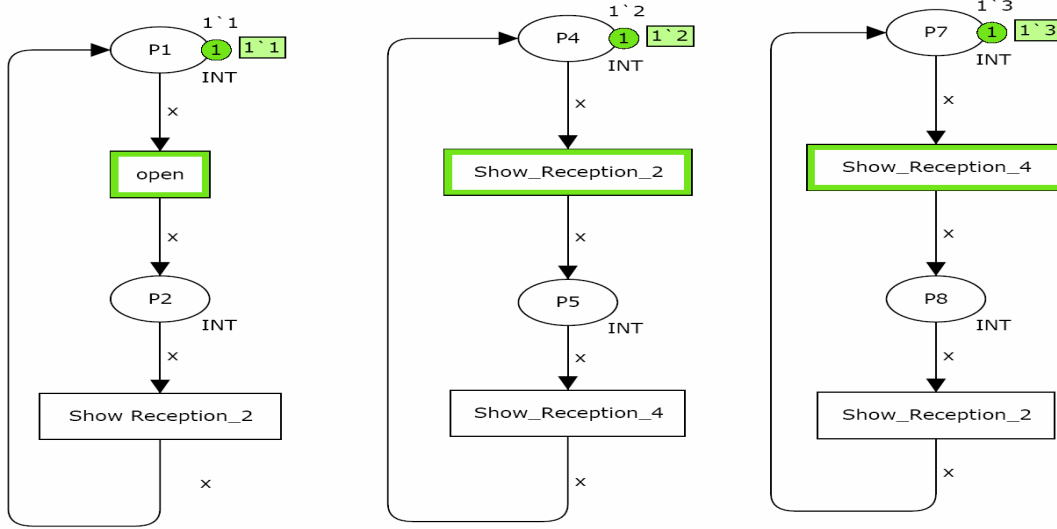


Figure 5-12 Intermediate CPN model of LSCs with loop example

The composition for more than two nets is performed with an assumption that associativity property holds for the composition rule defined till now. Hence the composition is performed by composing the first two charts and the resulting chart is composed with the third chart. For this example, composing either of the two charts first will give the same result as shown in Figure 5-13.

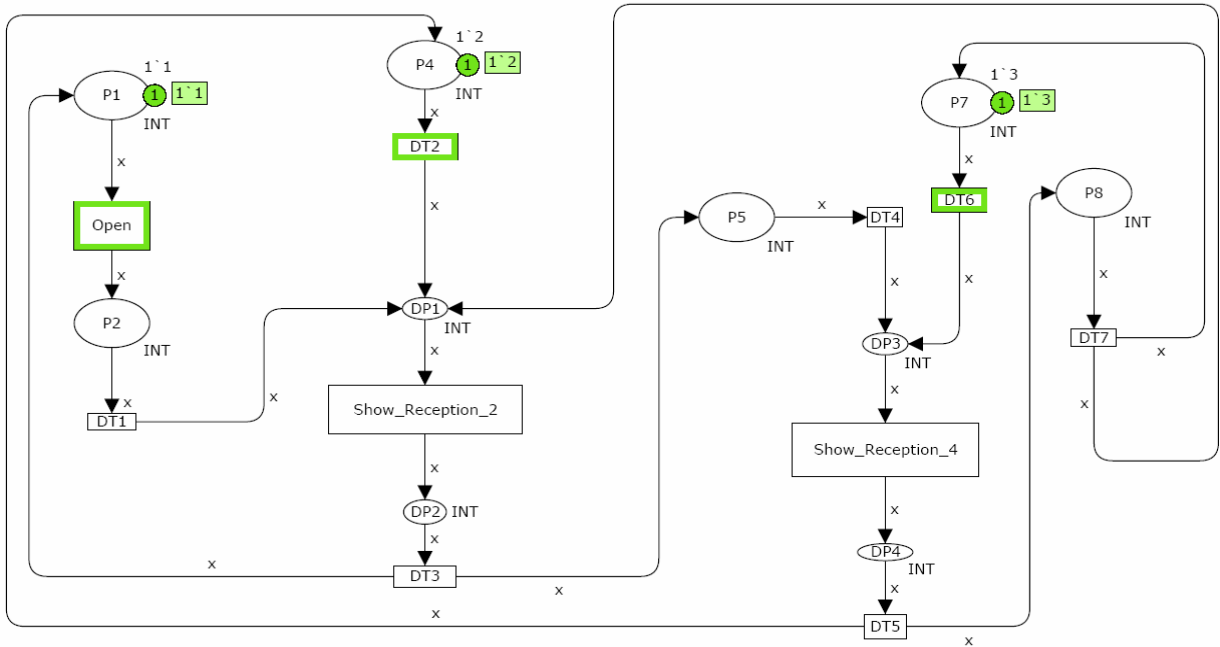


Figure 5-13 Transformed CPN model of LSCs with loop example

This composed model was analyzed using CPN Tools to perform the state space analysis and check different properties of Colored Petri Nets [9] and the summary of result of this analysis is

shown below. This result does not contain any abnormal result, except that that report is generated from partial state space analysis because of the infinite run of the system.

CPN Tools state space report for:
D:\LSC2CPN\work\CPN Tool\Composition of Petri Nets\Composition.cpn
Report generated: Sun Nov 26 08:48:31 2006

Statistics

State Space
Nodes: 16142
Arcs: 55578
Secs: 300
Status: Partial

Scc Graph
Nodes: 16142
Arcs: 55578
Secs: 3

Boundedness Properties

Best Integer Bounds

	Upper	Lower
New_Page'DP1 1	4	0
New_Page'DP2 1	4	0
New_Page'DP3 1	4	0
New_Page'DP4 1	3	0
New_Page'P1 1	4	0
New_Page'P2 1	4	0
New_Page'P4 1	1	0
New_Page'P5 1	4	0
New_Page'P7 1	1	0
New_Page'P8 1	3	0

Best Upper Multi-set Bounds

New_Page'DP1 1	3`1++
3`2++	
2`3	
New_Page'DP2 1	2`1++
2`2++	
2`3	
New_Page'DP3 1	3`1++
3`2++	
3`3	
New_Page'DP4 1	3`1++
3`2++	
2`3	
New_Page'P1 1	3`1++
3`2++	
3`3	
New_Page'P2 1	2`1++
3`2++	
2`3	
New_Page'P4 1	1`2
New_Page'P5 1	4`1++
4`2++	
4`3	

```

New_Page'P7 1      1`3
New_Page'P8 1      2`1++
2`2++
2`3

```

```

Best Lower Multi-set Bounds
New_Page'DP1 1      empty
New_Page'DP2 1      empty
New_Page'DP3 1      empty
New_Page'DP4 1      empty
New_Page'P1 1      empty
New_Page'P2 1      empty
New_Page'P4 1      empty
New_Page'P5 1      empty
New_Page'P7 1      empty
New_Page'P8 1      empty

```

Home Properties

```

Home Markings
None

```

Liveness Properties

```

Dead Markings
4755 [16142,16141,16140,16139,16138,...]

Dead Transition Instances
None

Live Transition Instances
None

```

Fairness Properties

```

No infinite occurrence sequences.

```

5.2.5. LSCs with Inconsistency

Another Special case of Non-Disjoint Live Sequence Chart is categorized a new class of LSC. This model is also composed into a single Colored Petri Net model by using the four rules described earlier. However there is one additional concept during composition which was not discussed earlier. The dummy place DP3 is added to ensure that a token is placed after execution of Thermo1_Change_70 only if this transition has been triggered from place P6. Figure 5-14 & Figure 5-15 shows the intermediate net of the LSC model of Figure 5-6 and Petri Nets model of Figure 5-16 is the composed Colored Petri Nets.

This model has some inconsistencies which will be identified later from the transformed model. Inconsistency in this model can be identified Live Sequence Chart model by playing it out using

Play Engine. Play Engine allows user to perform set of operation and also provide a test run (by designing Existential LSC) to check whether or not the sequence of operations is accepted by the LSC model. This requires user to carefully design test cases such that all possible path of execution is covered by test cases, which is error prone for large systems. However, the transformed CPN model does not requires defining all possible paths manually. Since CPN Tools explore every possible path, identification of inconsistency is autonomous in case of CPN Tools.

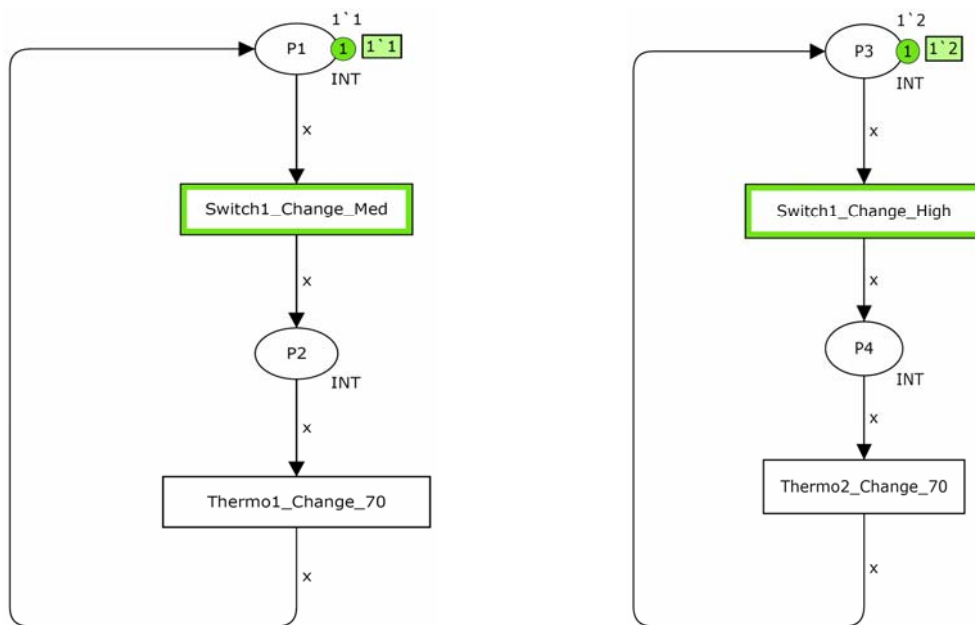


Figure 5-14 Intermediate CPN model of LSCs with inconsistency example

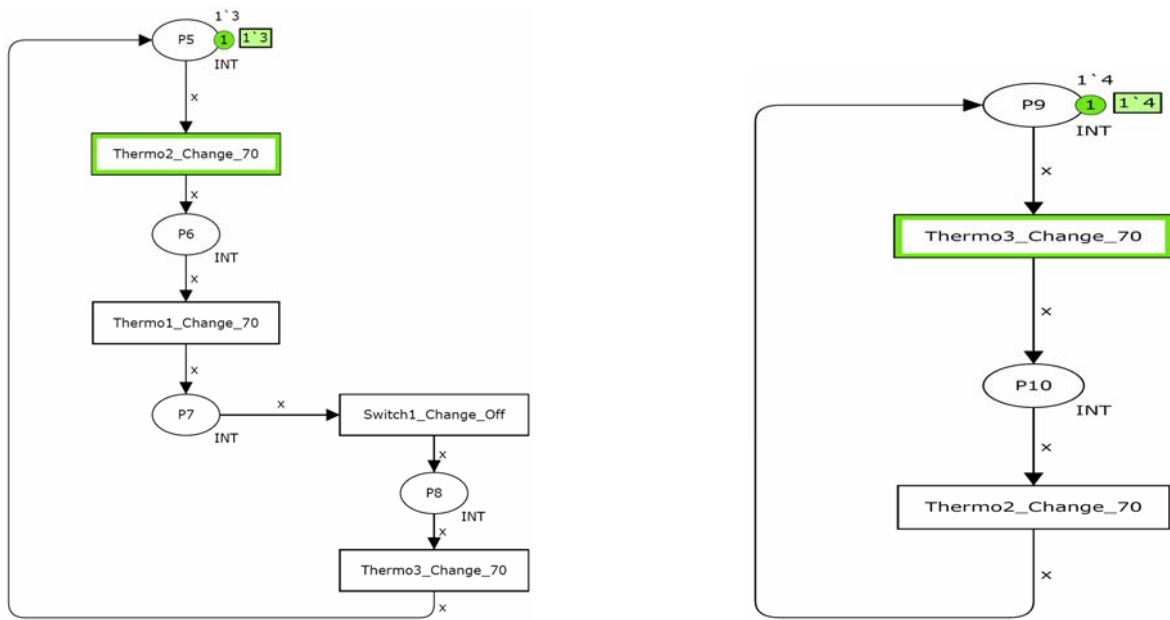


Figure 5-15 Intermediate CPN model of LSCs with inconsistency example

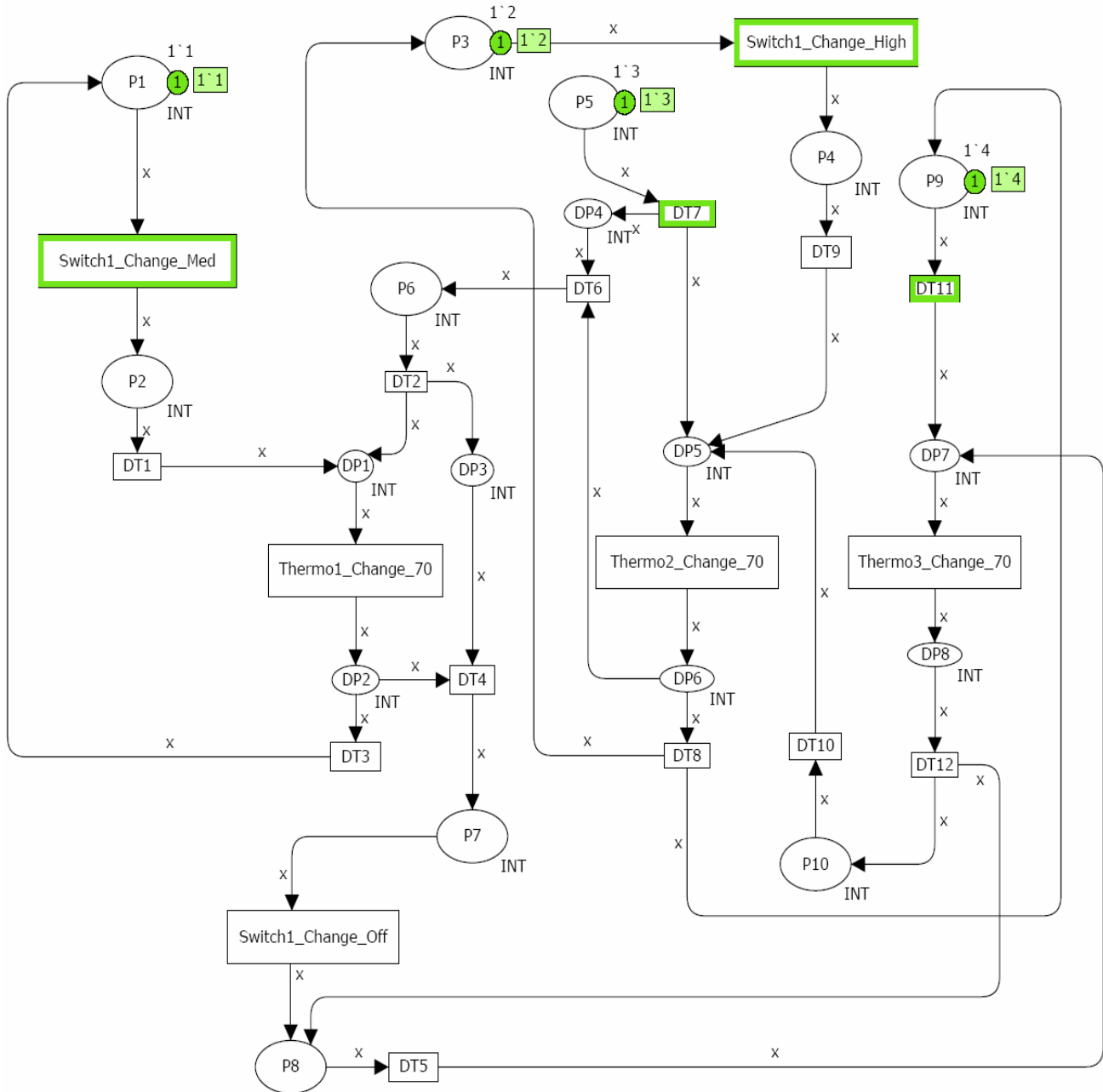


Figure 5-16 Transformed CPN model of LSCs with inconsistency example

6. Results

The transformed CPN model of each class is analyzed using CPN Tools. The state space analysis tool provided by CPN Tools produced a result showing Boundedness, Home, Liveness and Fairness properties [9]. The transformed model of case 5 (LSCs with inconsistencies) produced a remarkable result. A summary of result of the CPN Model of Figure 5-16 is shown at the end of this section.

In this state space analysis, the presence of dead transition instance signifies that there exists some transition which is never activated throughout the system run. This means the presence of such transition is not compulsory. After seeing the deadlock transition instance in analysis report of CPN model, we revisited the LSC model to see the cause of the problem. By performing a series of execution (Play-out) with the LSC model in Play Engine we were able to find out that the system violates LSC3 when switch1 is changed to Medium first and then from Medium to High. The reason for this violation is LSC3 states that once temperature of thermometer1 and thermometer2 is changed to 70, switch1 should be changed to off, however according to LSC2 when switch1 is changed to Medium, the state of switch1 should be in Medium. Because Switch1 cannot be in two states (Medium and Off) at the same time, hence the modeled system is inconsistent.

To check the inconsistency of system, we need to execute (play out) every path of execution in Play Engine. For larger systems often such kind of problem might be identified after a long system run. Thus manual simulation of such model becomes difficult. But the transformed model in CPN Tools has well defined tool for analysis which covers every path of execution and user has control over the model checking technique using CPN Tools by using Place Invariant and Transition invariant methods.

```
CPN Tools state space report for:
D:\LSC2CPN\work\CPN Tool\Composition Of Petri Nets\DeadlockRevisited.cpn
Report generated: Sun Nov 26 10:05:09 2006
```

Statistics

State Space

```
Nodes: 12082
Arcs: 62234
Secs: 300
Status: Partial
```

Scc Graph

```
Nodes: 5234
Arcs: 35110
Secs: 5
```


Boundedness Properties

Best Integer Bounds

	Upper	Lower
Compose'DP1 1	2	0
Compose'DP2 1	2	0
Compose'DP3 1	1	0
Compose'DP4 1	1	0
Compose'DP5 1	2	0
Compose'DP6 1	2	0
Compose'DP7 1	2	0
Compose'DP8 1	2	0
Compose'P10 1	0	0
Compose'P1 1	2	0
Compose'P2 1	2	0
Compose'P3 1	2	0
Compose'P4 1	2	0
Compose'P5 1	1	0
Compose'P6 1	1	0
Compose'P7 1	1	0
Compose'P9 1	3	0
Four'P10 1	1	0
Four'P9 1	1	0
One'P1 1	1	0
One'P2 1	1	0
Three'P5 1	1	0
Three'P6 1	1	0
Three'P7 1	1	0
Three'P8 1	1	0
Two'P3 1	1	0
Two'P4 1	1	0

Best Upper Multi-set Bounds

	Compose'DP1 1	1`1++
1`3		
	Compose'DP2 1	1`1++
1`3		
	Compose'DP3 1	1`3
	Compose'DP4 1	1`3
	Compose'DP5 1	1`2++
1`3		
	Compose'DP6 1	1`2++
1`3		
	Compose'DP7 1	1`2++
2`3++		
1`4		
	Compose'DP8 1	1`2++
1`3++		
1`4		
	Compose'P10 1	empty
	Compose'P1 1	1`1++
1`3		
	Compose'P2 1	1`1++
1`3		
	Compose'P3 1	1`2++
1`3		
	Compose'P4 1	1`2++
1`3		
	Compose'P5 1	1`3
	Compose'P6 1	1`3

Compose'P7 1	1`3
Compose'P9 1	2`2++
2`3++	
1`4	
Four'P10 1	1`4
Four'P9 1	1`4
One'P1 1	1`1
One'P2 1	1`1
Three'P5 1	1`3
Three'P6 1	1`3
Three'P7 1	1`3
Three'P8 1	1`3
Two'P3 1	1`2
Two'P4 1	1`2

Best Lower Multi-set Bounds

Compose'DP1 1	empty
Compose'DP2 1	empty
Compose'DP3 1	empty
Compose'DP4 1	empty
Compose'DP5 1	empty
Compose'DP6 1	empty
Compose'DP7 1	empty
Compose'DP8 1	empty
Compose'P10 1	empty
Compose'P1 1	empty
Compose'P2 1	empty
Compose'P3 1	empty
Compose'P4 1	empty
Compose'P5 1	empty
Compose'P6 1	empty
Compose'P7 1	empty
Compose'P9 1	empty
Four'P10 1	empty
Four'P9 1	empty
One'P1 1	empty
One'P2 1	empty
Three'P5 1	empty
Three'P6 1	empty
Three'P7 1	empty
Three'P8 1	empty
Two'P3 1	empty
Two'P4 1	empty

Home Properties

Home Markings
None

Liveness Properties

Dead Markings
4606 [9999,9998,9997,9996,9995,...]

Dead Transition Instances
Compose'DT10 1

Live Transition Instances
None

Fairness Properties

Compose'DT1 1	No Fairness
Compose'DT10 1	Fair
Compose'DT11 1	No Fairness
Compose'DT2 1	No Fairness
Compose'DT3 1	No Fairness
Compose'DT4 1	No Fairness
Compose'DT6 1	No Fairness
Compose'DT7 1	No Fairness
Compose'DT8 1	No Fairness
Compose'DT9 1	No Fairness
Compose'Switch1_Change_High 1	No Fairness
Compose'Switch1_Change_Med 1	No Fairness
Compose'Thermo1_Change_70 1	No Fairness
Compose'Thermo2_Change_70 1	No Fairness
Compose'Thermo3_Change_70 1	No Fairness
Four'Thermo2_Change_70 1	No Fairness
Four'Thermo3_Change_70 1	No Fairness
One'Switch1_Change_Med 1	No Fairness
One'Thermo1_Change_70 1	No Fairness
Three'Switch1_Change_Off 1	No Fairness
Three'Thermo1_Change_70 1	No Fairness
Three'Thermo2_Change_70 1	No Fairness
Three'Thermo3_Change_70 1	No Fairness
Two'Switch1_Change_High 1	No Fairness
Two'Thermo2_Change_70 1	No Fairness

7. Conclusions

As a result of the research in transforming a model drawn using Live Sequence Charts into Colored Petri Nets Model, we have built a prototype that gives a practical base of the results achieved so far. As a result we can see that the transformed model has some advantage over its source model where model checking can be performed more easily than in the source LSC model. Hence the result that can be achieved from exhaustive simulation in Play Engine can now be achieved in CPN Tools by automatic model checking.

The composition of the intermediate Petri Net model is done by combining two intermediate nets at a time. This incremental approach has a benefit that when a new LSC scenario is added in the source model, then the transformed CPN model can be simply composed of the intermediate Petri Net model of the new scenario. Thus addition of new scenario does not require re-generation of an entire CPN model, instead an incremental change with existing CPN model should work correctly.

8. Future Plans

The prototype of LSCTOCPN has been developed to provide experimental results. Since the tool used to model the Live Sequence Chart is also a research prototype, enhancement in this tool (Play Engine) will highly affect the working mechanism of my product (LSCTOCPN). However the architecture of LSCTOCPN has flexibility to adapt the newer version of both Play Engine file as well as CPN Tools.

The Transformation rules discussed in Section 5 covers a subset of the LSC domain and there are still more cases which needs to be considered in the transformation rules. As part of my master's thesis, I will be extending the work of LSCTOCPN with the following two objectives:

- ❑ Generalization of the Transformation Rules: The Current Version of LSCTOCPN tool supports for the five different classification of LSC model discussed earlier. Further exploration needs to be done to find all the possible classifications of LSC constructs and to come up with their transformation rules. A general approach for the identification of different LSC models into their corresponding classification should be figured out so that appropriate transformation rule can be applied to a particular model.
- ❑ Formalization of the Composition Rules for the Live Sequence Charts: A theoretical approach in formalization of Composition Rules for the Live Sequence Charts.

The current version of Play Engine tool does not support all the LSC constructs that have been identified in the literature written after the development of initial version of Play Engine. Hence LSCTOCPN tool has to depend on the new version of Play Engine which will consider new definitions of the Live Sequence Chart.

As part of my master's thesis if the newer classification of LSC model is not supported by the latest version of Play Engine, then a plug-in for the Play Engine or a separate tool to support modeling these constructs need to be developed.

9. Appendix

9.1. LSCTOCPN Tool

LSCTOCPN takes as an input the workspace file of Play Engine and generates the output file that can be readable by CPN Tools. The input Workspace file of play engine is parsed by the PlayEngine Package. In this process, the features of Use Cases represented using Live Sequence Charts are internally stored in the data structure (PlayEngine.LSCWorkSpace class) as a source object for the conversion process. More detail about this data structure is discussed in the next section below. The input object of class PlayEngine.LSCWorkSpace will be then fed into UMASSD.LSCTOCPN Package to extract its equivalent CPN Model. This package uses the transformation rules discussed above to build an equivalent Colored Petri Nets Model which will be internally stored into another data structure (CPNTOOLS.CPNWorkSpace class). This data structure will be used by CPNTools package to produce the output file (.cpn). The structure of output file is explained in Section 9.1.1 - 9.1.3.

The architecture of LSCTOCPN is presented in Figure 9-1

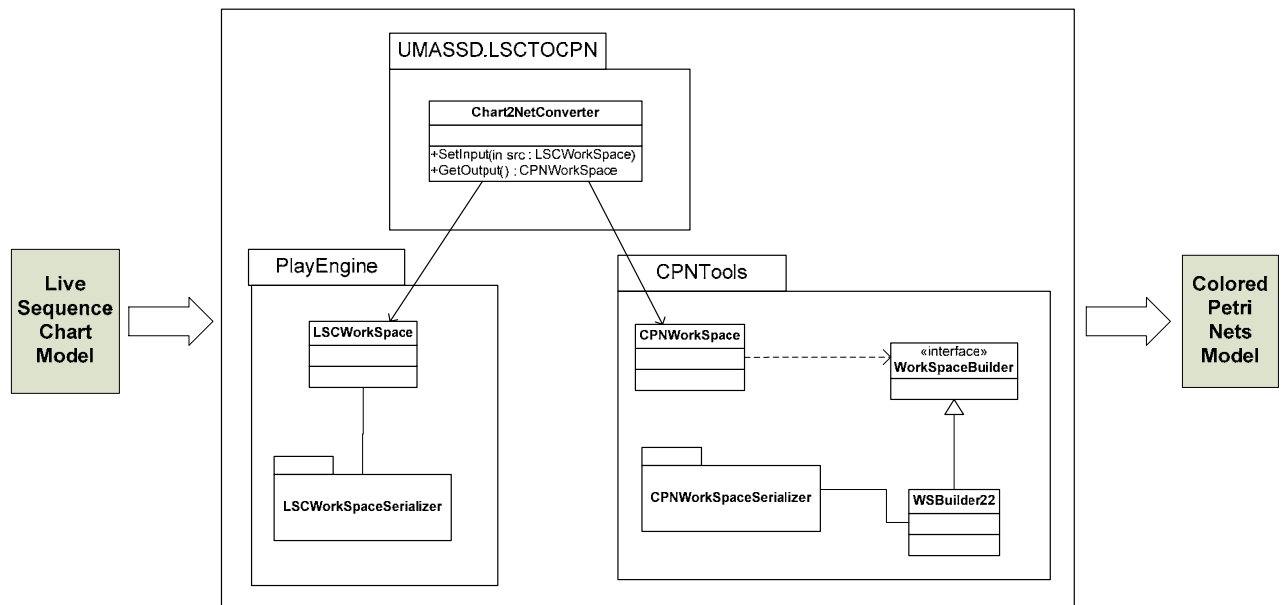


Figure 9-1 Architecture of LSCTOCPN tool

9.1.1. Package PlayEngine

PlayEngine package consists of two major classes LSCWorkSpace and LSCWorkSpaceSerializer. LSCWorkSpace is a container class which stores all the information gathered from the source file. LSCWorkSpaceSerializer is responsible for parsing the source workspace file of PlayEngine and extract the required information.

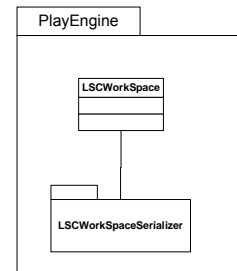


Figure 9-2 Package PlayEngine

9.1.2. Package CPNTools

CPNTools package contains four major components CPNWorkspace, CPNWorkSpaceSerializer and WSBUILDER22 and WorkSpaceBuilder. Class CPNWorkspace is a storage class which stores all the results converted from the object responsible for conversion from Live Sequence Chart to Colored Petri Nets. This Class will be serialized to a file (.cpn) readable by CPN Tools. WSBUILDER22 is a class responsible for saving .cpn file compatible with CPN Tools Version 2.2.0.

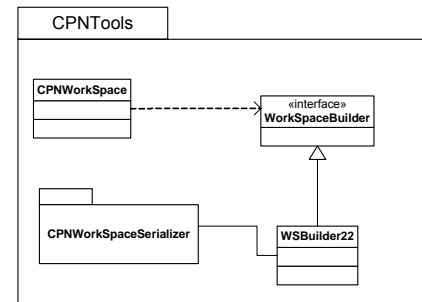


Figure 9-3 Package CPNTools

9.1.3. Package UMASSD.LSCTOCPN

This package has one Class Chart2NetConverter which is responsible for converting objects of class PlayEngine.LSWWorkSpace to its equivalent class CPNTools.CPNWorkspace. This corresponding CPNWorkspace model is created when “SetInput (...)” method of this object is invoked with instance of LSCWorkspace as its parameter. The conversion process uses the rules defined in Section 5.

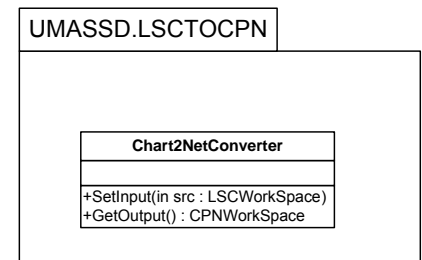


Figure 9-4 Package UMASSD.LSCTOCPN

9.2. PlayEngine Workspace Schema

The workspace file created using Play Engine consists of five different files:

- Workspace (xyz.ews)
- Application (Appxyz.xml)
- Application Extension (xyz.ext)
- Specification (Specxyz.xml)
- GUI Application DLL (xyz.dll)

Except the dll file, all files are using xml format to represent its content. These files do not have any external Document Type Definition (DTD) or XML Schema Definition (XSD) file. As part of implementation of this project, the first step was to identify or create metadata information of these file structures.

Various tools were used to automatically extract the XML Schema: Microsoft XSD inference [34], XML Utilities from HiTSoftware [35] and Altova XML Spy [36]. All of them have some advantage over the other but eventually Microsoft XSD inference was used since the output of XSD inference was compatible with XSD Object Code Generator downloaded from [34] after some manual modification in the generated XSD file. XSD Object Code Generator can be used to generate a serializer class (in C#) for XML files.

Hence the first module of LSCTOCPN is generated using XSD Object Code Generator and Microsoft XSD inference. These generated classes are defined inside the namespace "PlayEngine". An additional class "LSCWorkSpace" has been defined to store the extracted information that can be used by the class responsible for mapping (Package UMASSD.LSCTOCPN).

9.2.1. Workspace (.ews)

This is the main workspace file which stores the information of all related files for Play Engine Tool, and the configuration settings while loading a workspace file in PlayEngine.

9.2.2. Application (Appxyz.xml)

This file stores information about all the data types and objects that are used in the application. This file is also internally stored in XML file format.

9.2.3. Application Extension File (xyz.ext)

This file stores the information about the GUI Objects created using Visual Basic and the list of states of each enumerations which will be associated with these GUI Objects.

9.2.4. Specification File (Specxyz.xml)

This is the main file which stores all the use cases of Live Sequence Charts. A set of Live Sequence Charts are defined under a single use cases and this specification file can represent multiple use cases. Both Universal LSC and Existential LSCs are defined in this file.

9.3. CPN Tools Schema

CPN Tools [31] uses a single file to store all the information about its pages and the Colored Petri Nets model inside each of those pages. With the file extension (.cpn), the format of this file is in XML representation. Similar to Play Engine serializer, we require a serializer package for CPN

Tools. Developers of CPN Tools have defined a document type definition which is used to validate a .cpn file. Thus the DTD [37] file was downloaded from [38] which was then converted to XML Schema Definition (XSD) [39] using Altova XML Spy [36]. Again for CPN Tools a serializer class (in C#) for .cpn file is generated using XSD Object Code Generator [34]. All generated classes are packaged inside CPNWorkspaceSerializer module. LSCTOCPN tool will store all the results into CPNWorkspace class, thus the information contained inside CPNWorkspace will be saved into .cpn file using CPNWorkspaceSerializer and WorkspaceBuilder.

10. References

- [1] Harel, D. "From Play-In Scenarios to Code: An Achievable Dream" (2001)
- [2] Brill, M., Damm, W., Klose, J., Westphal, B., & Wittke, H., "Live Sequence Charts an introduction to lines, arrows and strange boxes in the context of formal verification", Lecture notes in computer science, Springer, 374-399
- [3] ITU-T: ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU-T, Geneva (1993)
- [4] ITU-T: ITU-T Annex B to Recommendation Z.120: Algebraic Semantics of Message Sequence Charts. ITU-T, Geneva (1995)
- [5] Bontemps, Y. & Heymans, P., Turning High-Level Live Sequence Charts into Automata *Univ. of Namur - Computer Science Dept*, **2002**
- [6] Bontemps, Y. & Heymans, P., Turning High-Level Live Sequence Charts into Automata Proc. of "Scenarios and State-Machines: models, algorithms and tools" (SCESM) Workshop of the 24th Int. Conf. on Software Engineering (ICSE 2002), **2002**
- [7] Bontemps, Y., Heymans, P. and Kugler, H. Applying LSCs to an Air Traffic Control Case Study , May 2003, 2nd international workshop on Scenarios and State Machines: Algorithms, State Machines and Tools (SCESM'03), Portland, OR.
- [8] Lorentsen, L., Tuovinen, A. & Xu, J. "Modelling of Feature Interactions in Nokia Mobile Phones using Colored Petri Nets", Proceedings of Application and Theory of Petri Nets, 2002.
- [9] Jensen, K. "Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use" (Volume 1, 2 & 3), Springer-Verlag, 1992
- [10] Bernardi, S., Donatelli, S. & Merseguer, J. "From UML Sequence Diagrams and State Charts to Analysable Petri Net Models", Workshop on Software and Performance, 2002
- [11] Hu, Z. & Shatz, S. M. "Mapping UML Diagrams to a Petri Net Notation for System Simulation", In Proc. Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE'04), 2004
- [12] Wilkerson, B., Wirfs Brook, R. & Wiener, L. "Designing Object Oriented Software", Prentice Hall PTR, 1990
- [13] Bastide, R. "Approaches in Unifying Petri Nets and the Object Oriented Approach", Proceedings of the Application and Theory of Petri Nets, 1995
- [14] Bauskar, B., "Integration of object oriented design and colored Petri Nets using abstract node approach", Masters Thesis, Department of Computer Science, University of Massachusetts Dartmouth, 2004.
- [15] Amorim, L.; Maciel, P., Nogueira, M., Barreto, R. & Tavares, E. "Mapping Live Sequence Charts to Coloured Petri Nets for analysis and verification of embedded systems", SIGSOFT Softw. Eng. Notes, ACM Press, 2006, 31, 1-25

- [16] Damm, W. & Harel, D. "LSCs: Breathing Life into Message Sequence Charts Formal Methods in System Design", Springer, 2001, 19, 45-80
- [17] Van Eijk, C.A.J. "Sequential equivalence checking without state space traversal", IEEE Computer Society, 1998, 618-623
- [18] Girault, C. & Valk, R. "Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications", Springer-Verlag New York, Inc., 2001
- [19] Harel, D., Kugler, H. & Pnueli, A. "Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements", Springer, 2005, 309-324
- [20] Sibertin-Blanc, C. "A Client-Server Protocol for the Composition of Petri Nets", 1993, 377-396
- [21] Sun, J. & Dong, J.S. "Design Synthesis from Interaction and State-Based Specifications", IEEE Trans. Software Eng., 2006, 32, 349-364
- [22] Marsan, M.A., Balbo, G., Conte, G., Donatelli, S. & Franceschinis, G., "Modeling with Generalized Stochastic Petri Nets", J. Wiley, 1995
- [23] Ratzer, A.V. et. al. "CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets", Dept. of Computer Science, University of Aarhus.
- [24] Harel, D. & Marelly, R. "Come Let's Play: Scenario Based Programming Using LSCs and the Play-Engine", Springer Verlag (2003)
- [25] Mcleish, K. "Petri Nets" [<http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C10/10-7.html>]
- [26] Rational Rose from Answers.com [<http://www.answers.com/topic/rational-rose>]
- [27] Rhapsody Software [<http://www.ilogix.com/sublevel.aspx?id=53>]
- [28] Petri Nets Tools Database Quick Overview
[<http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html>]
- [29] CPN Tools-Help [<http://wiki.daimi.au.dk:8000/cpntools-help/cpntools-help.wiki>]
- [30] Reniers, M. A., Introduction to Message Sequence Chart
[<http://www.win.tue.nl/~michelr/mscintro/mscintro.html>], 1992
- [31] CPN Tools Wiki [<http://wiki.daimi.au.dk/cpntools/cpntools.wiki>]
- [32] MAFIA Project [<http://www.daimi.au.dk/CPnets/MAFIA/>]
- [33] HP Projects [<http://www.daimi.au.dk/CPnets/HP/>]
- [34] Microsoft XSD Inference 1.0 [<http://www.gotdotnet.com/team/xmltools/>]
- [35] XML Tools [http://www.hitsw.com/xml_utilites/]
- [36] Altova XML Spy [<http://www.altova.com/>]
- [37] DTD @ W3Schools [<http://www.w3schools.com/dtd/default.asp>]
- [38] CPN Tools DTD [<http://www.daimi.au.dk/~cpntools/bin/DTD/6/cpn.dtd>]
- [39] XML Schema [<http://www.w3.org/XML/Schema>]
- [40] [<http://www.microsoft.com/downloads/details.aspx?familyid=89e6b1e5-f66c-4a4d-933b-46222bb01eb0&displaylang=en>]
- [41] CPN Tools + UML @ NOKIA [<http://www.daimi.au.dk/CPnets/UMLCPNatNokia/>]