Integrating UML Statechart and Collaboration Diagrams Using Hierarchical Predicate Transition Nets

Zhijiang Dong, Xudong He

School of Computer Science
Florida International University
Miami, FL 33199, U.S.A.

Abstract: UML has become the standard object-oriented analysis and design language in software industry. However UML lacks a precise semantics that hinders error detection in the early stages of system development. Even worse, there is no clear definition of consistency criteria among various UML notations, and there are few examples of illustrating the use of various notations together. In this paper, we present an approach of using Hierarchical Predicate Transition Nets (HPrTNs) to define and integrate UML statechart diagrams and collaboration diagrams. Our approach establishes a basis for relating various UML models together and for carrying out formal analysis.

1 Introduction

UML [BRJ99], developed by a group of leading experts in object-oriented anaylsis and design, has become the standard object-oriented development methodology in software industry. Its usefulness and impact on software development will be extremely profound in the coming years. However, many graphical notations in UML only have informal English definitions [UML99] and thus are error-prone and cannot be formally analyzed. Furthermore, there is no clear definition of consistency criteria among various UML diagrams. As a result, it is difficult to know how to use different UML diagrams together to model a system.

With the goal to better understand UML and to reveal potential problems in the current definition of UML and to eventually formally analyze UML specifications and designs, many researchers are currently trying to formally define the meanings of UML notations. Our research shares the same goals of other researchers. Although there are many existing works on integrating different formal methods with object-oriented methods, research work on applying formal methods to UML specific notations is still rare. Of course one of the major reasons is that the UML specification documents of the OMG (Object Management Group) are still under constant revisions.

The chosen formalism of our study is hierarchical predicate transition nets (HPrTNs), which are a class of hierarchical high-level Petri nets developed in [He96]. HPrTNs are graphical and executable and are especially suited for specifying and analyzing concurrent and distributed systems. Due to their graphical and executable nature, HPrTNs can be used to formally define many other UML notations dealing with behavior models. We have developed a general approach to define object-oriented concepts using HPrTNs [HD01]. We have applied HPrTNs in defining UML class diagrams [He00a,b] and use case diagrams [He00c].

This paper extends our earlier results in formalizing UML diagrams. In this paper, we present an approach to derive and synthesize HPrTNs from a system description modeled by statechart diagrams and collaboration diagrams. More specifically, we provide (1) a method to derive a state Petri net model (SPNM) from a statechart diagram, and (2) a method to synthesize derived SPNMs according to collaboration diagrams. Our results establish the foundation for formally analyzing the dynamic behavior and properties of UML diagrams and provide a way for checking the consistencies between several UML diagrams.

2 Related Work

In recent years, there have been considerable research activities in the area of defining more precise semantics of UML notations. There is a worldwide pUML (precise UML - http://www.cs.york.ac.uk/puml/) research group and there is an annual UML conference, initiated in 1998.

Researchers have attempted to define formal semantics for class diagrams ([SF97], [Eva98], [FEL98], [LB98], [KC99], [He00a], [He00b]), use-case diagrams ([OP98], [BPP99], [He00c]), and interaction diagrams ([Kna99], [SS00]). A more comprehensive overview of recent results can be found at http://ctp.di.fct.unl.pt/~amm/wrk14.html, for the Workshop on Defining Precise Semantics of UML in the European Conference on Object-Oriented Programming 2000. The chosen formal methods used in the above works include:

(1) variants of Z [Spi92]: [SF97], [Eva98], [KC99];
(2) variants of logic: [OP98], [Kna99];
(3) refinement calculus: [BPP99];
(4) variants of Petri nets: [He00a], [He00b], [He00c], [SS00].

It can be seen that there is no a widely accepted formalism for defining UML and there is no a unified formalization for multiple UML notations, which is important for practical problems. Among the various formal methods used, variants of Petri nets have been applied to defining class diagrams ([He00a], [He00b]), use case diagrams [He00c], and startcharts and collaboration diagrams [SS00]. In addition, a recent volume (2001) of lecture notes in computer science [ADR01] contains a variety of approaches in using Petri nets to model OO concepts.

The work most closely related to ours is [SS00]. [SS00] used object Petri nets [Lak01] in modeling UML statecharts and collaboration diagrams. Their work provided some useful ideas on handling events in translating statechart diagrams; however, there are several major differences between their work and ours. First, we translate a statechart diagram directly into an HPrTN that maintains the hierarchical structural information of the original statechart diagram while the statecharts had to be flattened before they could be translated in [SS00]. Maintaining structural information not only help pinpoint to the possible defects in the UML specification but also facilitates potential code generation and maintenance. Second, our approach supports both synchronous and asynchronous integrations of derived Petri net models according to the collaboration diagrams while only synchronous integration was considered in [SS00]. Furthermore, we provide much more detailed translation rules than [SS00].

3 Hierarchical Predicate Transition Nets

A hierarchical predicate transition net (HPrTN) is a predicate transition net [Mur89] with hierarchies for system structure modeling [He96]. An HPrTN *N* consists of (1) a finite hierarchical net structure (P, T, F, ρ), (2) an algebraic specification SPEC, and (3) a net inscription (φ, L, R, M_0).

(P, T, F) is the essential net structure, where $P \cup T$ is the set of nodes satisfying the condition $P \cap T = \emptyset$. P is called the set of places and T is called the set of transitions. There are two kinds of nodes for both places and transitions - elementary nodes (represented by solid circles or boxes) and super nodes (represented by dotted circles or boxes). Elementary nodes have the traditional meaning in flat Petri net models. Super nodes are introduced to abstract and refine data and processing in HPrTNs [HL91]. F is a set of arcs denoting flow relationships. An arc in an HPrTN may represent a cluster of flows due to the use of super nodes. $\rho: P \cup T \to \wp(P \cup T)$ is a hierarchical mapping that defines the hierarchical relationships among the nodes in P and T. The underlying specification SPEC is a meta-language to define the tokens, labels, and constraints of an HPrTN. Arc labels can contain label constructor + for non-deterministic flows and × for concurrent flows. The net inscription (φ, L, R, M_0) associates each graphical symbol of the net structure (P, T, F, φ) with an entity in the underlying SPEC, and thus defines the static semantics of an HPrTN. The dynamic semantics of an HPrTN is defined in terms of its flattened PrTN. Due to the space limit, the details of HPrTN are omitted in this paper and can be found in [He96].

4 An Approach to Integrate Statechart and Collaboration Diagrams in HPrTNs

In this section, we present our translation approach, which consists of

(1) a method to derive a state Petri net model (SPNM) from a statechart diagram, and
(2) a method to obtain a system model (SM) by integrating derived SPNMs according to collaboration diagrams.

4.1 Generating State Petri Net Model (SPNM)

In UML, a statechart diagram is used to define the dynamic behaviors of the objects of a class by showing state changes responding to events [OMG00].

In the following sections, we show how to formally define UML statechart diagram using HPrTNs incrementally. To simplify the discussion, only necessary net components and thus incomplete HPrTNs are used to illustrate relevant features.

4.1.1 Representing States

A state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action or waits for some event. There are two kinds of states: simple states and composite states.

A simple state may be subdivided into multiple compartments separated from each other. They are name compartment and internal transition compartment. Each internal transition compartment can hold a list of internal actions or activities that are performed while the element is in the state. There are two reserved action labels – entry and exit.

An event is a noteworthy occurrence while a message is a specification of stimulus [OMG00], which is a communication between two objects that conveys information with the expectation that some action will ensue. Both of the above concepts can be denoted by tokens in HPrTNs.

Rule 1 – Event Rule: In SPNM, each event is represented using a token of the type: <Sender, Receiver, Return_transition, Event_type, Event_name, Event_signature>.

Sender and Receiver denoting the source and target of an event respectively are of type OBJ_ID, which is the type of object identifications in the SPNM. Receiver can be null, which means every object that is interested in the event can receive this event. Return_transition, used for synchronization, is the name of a transition that requires the reply of this event as an input. Return_transition can be null, which means Receiver does not need to reply this event. Event_type indicates the type of the event. Without an event type, there can be some problems. For example, if all objects of class B is interested in event E generated by an object of A, we may have several different situations: (1) If one object of class B consumes E, then other objects cannot get E; (2) if none of the objects of class B consumes E, then E will stay forever in the "System Event Center" to be discussed later. In order to solve these problems, we define the following event types:

- FOREVER: A copy of the event stays in System Event Center forever.
- INSTANT: Whenever an object receives the event from "System Event Center", "System Event Center" cannot keep the event at the same time. This means the event is consumed.
- MESSAGE: The event is a type of message and a corresponding operation is invoked. It is reserved for future use.
- NOTIFY (obj_id.Event_name): the "System Event Center" keeps a copy of the event until the specified event is generated by the designated obj_id. Obj_id can be a class name, which means if "System Event Center" receives the specified event from any object of the class, the current event in "System Event Center" is consumed at the same time.
- REPLY: The event is a reply for some event. In this case, the Return_transition cannot be null and Receiver cannot be null or be a class name.
- TIMES (n): whenever a object receives the event from "System Event Center", the event count in "System Event Center" becomes TIMES(n-1). If n-1 is equal to 0, the "System Event Center" consumes the copy.

Event_name and Event_signature contain information of the event, the latter indicates the event parameter type and value information.

Rule 2 – Simple State Rule: A simple state is represented by a super place, consisting of the following components:

- An EntryPoint place, denoting the entry of the current state;
- An EntryAction transition, sending corresponding event to OutEvent to invoke the entry actions when fired;
- A KernelState place, denoting the current state without entry and exit actions;

- An ExitAction transition, sending corresponding event to OutEvent to invoke the exit actions when fired;
- An ExitPoint place, denoting the exit of the current state;
- An ActionMSG place, used to store the messages that should be invoked when entry or exit the state.

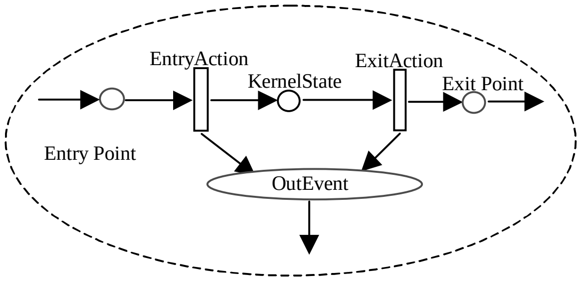

Fig. 1 An HPrTN Definition of A Simple State

If a state does not have the Entry/Exit actions, then EntryPoint/ExitPoint, EntryAction/ ExitAction and OutEvent are not needed. In this situation, KernelState is enough to represent the state.

A composite state is decomposed into concurrent substates or mutually exclusive disjoint substates. A given state may only be refined in one of these two ways. Naturally, any substate of a composite can also be a composite state of either type.

Rule 3 – Disjoint Composite State Rule: A composite state with disjoint sub states is represented by a super place, consisting of following components:

- EntryPoint place, the same as these in Rule 2;
- EntryAction transition, the same as these in Rule 2;
- ExitPoint place, the same as these in Rule 2;
- ExitAction transition, the same as these in Rule 2;
- OutEvent place, the same as these in Rule 2;
- RealSPNM super place, a sub SPNM of the sub statechart diagram.

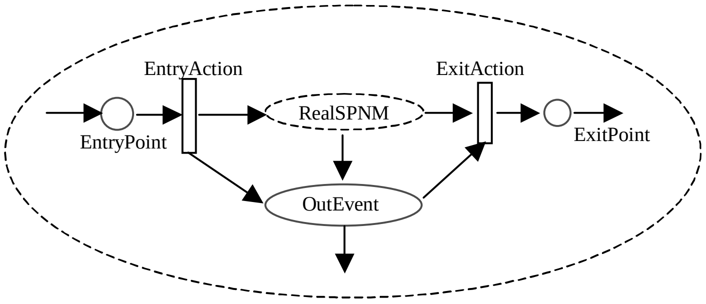

Fig. 2: An HPrTN Definition of A Disjoint Composite State

Rule 4 –Concurrent Composite State Rule: A composite state with concurrent sub states is represented by a super place, consisting of following components:

- EntryPoint, the same as the one in simple state;
- EntryAction, the same as the one in simple state;
- ExitPoint, the same as the one in simple state;
- ExitAction, the same as the one in simple state;
- ActionMSG, the same as the one in simple state;
- State_Path, a super place representing composite state of each path.

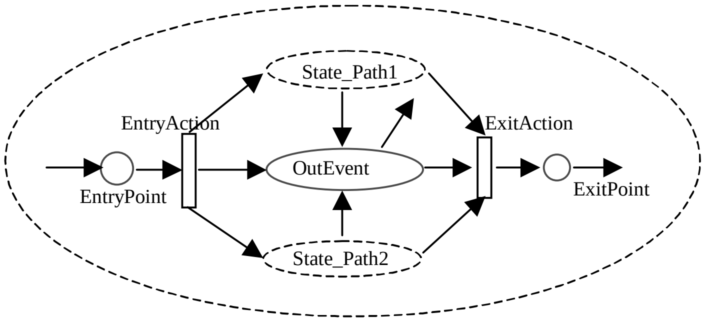

Fig. 3: An HPrTN Definition of A Concurrent Composite State

Composite state can have another property: the current behavior of the state depends on its past. A history composite state is allowed to remember the last substate that was active in it prior to the transition from the composite. Following rule is used to convert history composite state.

Rule 5 – History Composite State Rule: A composite state with history is realized in HPrTNs by adding a History place and a Distributor transition to the composite state. The History place becomes the only entry of the composite state.

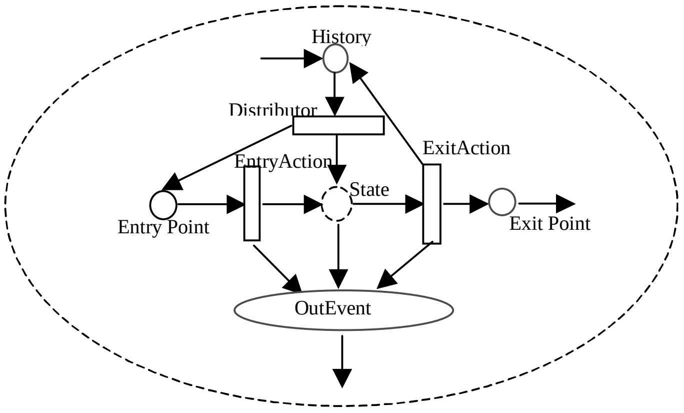

Fig. 4: An HPrTN Definition of A Composite State with Histroy

The History place records the last left state by firing ExitAction transition, so whenever reentered, it can get into the appropriate state by firing the Distributor transition.

4.1.2 Transitions

In a statechart diagram, a transition can have five parts: source state, event signature, guard condition, action expression and target state.

An action, represented by action expression, must be completed before entering into the target state. This means that synchronization needs be used in the HPrTN definition. The following rule gives the HPrTN definition of transition.

Rule 6 – Transition Rule: A transition in UML is represented by a super transition of HPrTN, consisting of the following components:

- A “GuardCondition” transition, whose constraint is defined by a guard condition;
- A “SynPoint” place;
- A “Leave” transition, waiting for the replies of the action which is represented by the events sent from “GuardCondition” to “OutEvent”.

When “GuardCondition” transition is fired, it sends the event(s) representing the action expression to “OutEvent”. Sometimes it also sends a corresponding event to “InEvent” and enters into “SynPoint” place waiting for the event(s) indicating the action expression has been finished. The action expression has been finished if the “Leave” transition is enabled. When “Leave” transition is fired and enters into the target state, the transition of UML is finished.

If a transition of in a statechart diagram does not include any action expression, “SynPoint” and “Leave” are not needed.

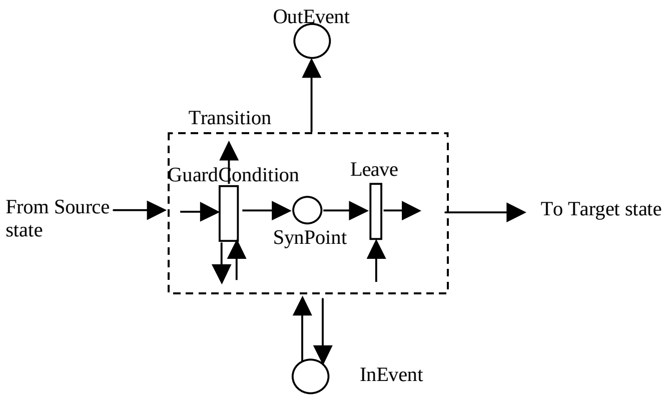

Fig. 5: An HPrTN Definition of A Transition

4.1.3 Statechart Diagram

The following rule describes how to translate a statechart diagram into an HPrTN.

Rule 7 – Statechart Diagram Rule: A statechart diagram is represented by a super place, consisting of the following components:

- A super transition "Transitions", consisting of internal places, derived from states of UML according to rules of section 4.1.1, and interface transitions, derived from the transitions of UML according to rules of section 4.1.2;
- An "InEvent" place, storing events relating with statechart diagram;
- An "OutEvent" place, storing events generated by statechart diagram;
- A "TimeMonitor" transition, used to generated events relating with time.

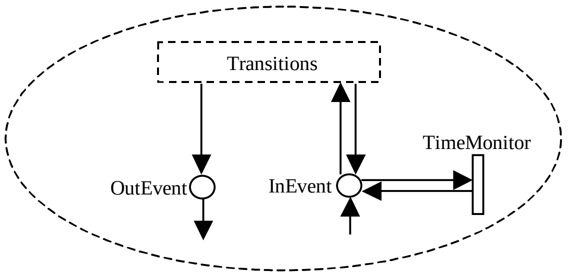

Fig. 6: An HPrTN Definition of Statechart Diagram

If there are no events with time, transition "TimeMonitor" can be omitted.

4.1.4 State Petri Net Model

From the above sections, we know how to convert a statechart diagram to a corresponding HPrTN. The following rule is used to obtain a SPNM.

Rule 8 – State Petri Net Model Rule: An SPNM is represented by a super transition, consisting of following components:

- A "Recv Event" transition, which receives events from its environment and sends the events to "InEvent" of statechart;
- A "Send Event" transition, which receives events from "OutEvent" of statechart and sends to its environment;
- A "StateChart" super place, representing the corresponding statechart diagram.

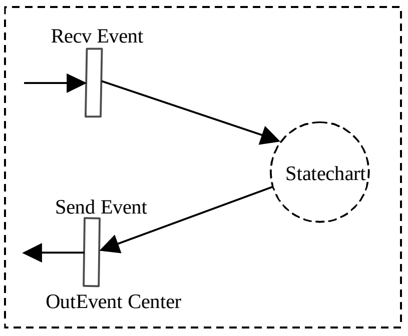

Fig. 7: An HPrTN Definition of SPNM

4.2 Generating System Model from UML Collaboration Diagrams

A collaboration diagram defines the interactions among objects of different classes. To define object interactions in HPrTNs, we use one place: System Event Center visible to its environment to store all events generated by different objects. If the system is open, it must need events from its environment.

The system model can be viewed as a larger HPrTN, so we can change the system model to another State Petri Net Model by applying the corresponding rules, which can form a larger system model with the environment of the current system model.

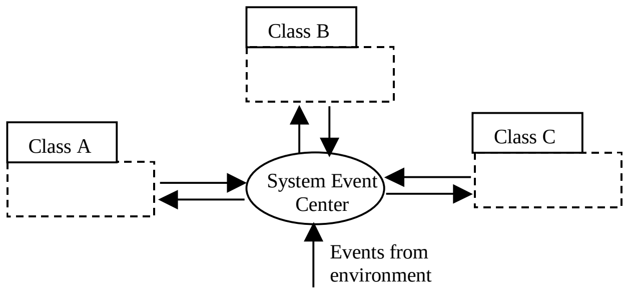

Fig. 8: An HPrTN Definition of Collaborations

Although the sequence of events can be obtained by executing the system model, it is convenient to define the sequence explicitly. We use the following arc labeling convention to describe the sequence of events: <sequence number, sender, receiver, event name, event parameters>. For example, if A sends event SetValue(d, 4) to B with sequence number 2.1.1 in UML collaboration diagram, then a label is added to the transition from class A to System Message Center: <2.1.1, A, B, SetValue, <d, 4> >.

5 A Translation Example

We use the following a one-minute microwave oven adapted from [SM92] to illustrate our translation approach. The microwave oven has the following points:

1) Only one control button for the user. If the button is pushed and the door of oven is closed, the oven will cook (that is energize the power tube) for 1 minute.
2) Push the button at any time when the oven is cooking, you get an additional minute of cooking time.
3) Pushing the button when the door is open has no effect.
4) There is a light inside the oven. Any time the oven is cooking or the door is open, the light must be turned on.
5) If the door is closed, the light goes out.
6) If the oven times out, it turns off both the power tube and the light. It then emits a warning beep to tell the user that the food is ready.

In this example, we define the following events. For microwave oven, there are five events: V1: Button pushed; V2: Timer timed; V3: Door opened; V4: Door closed; V5: Time Changed. Both light and power tube have two events respectively: L1: Turns on light, L2: Turns off light for light and P1: Energizes power tube, P2: De-energizes power tube for power tube. User can generate four events: U1: User is pushing button; U2: User is opening the door; U3: User is closing the door and U4: User is accessing the food. For an event V, let V' be the reply of event V.

The system model consists of the following essential UML diagrams: a class diagram showing four classes and their relationships, four statechart diagrams modeling the behaviors of the objects of the four classes respectively, and a collaboration diagram defining the dynamic interactions among the objects of the four classes. To illustrate our translation approach, we only show the statechart diagram for oven and the collaboration diagram here.

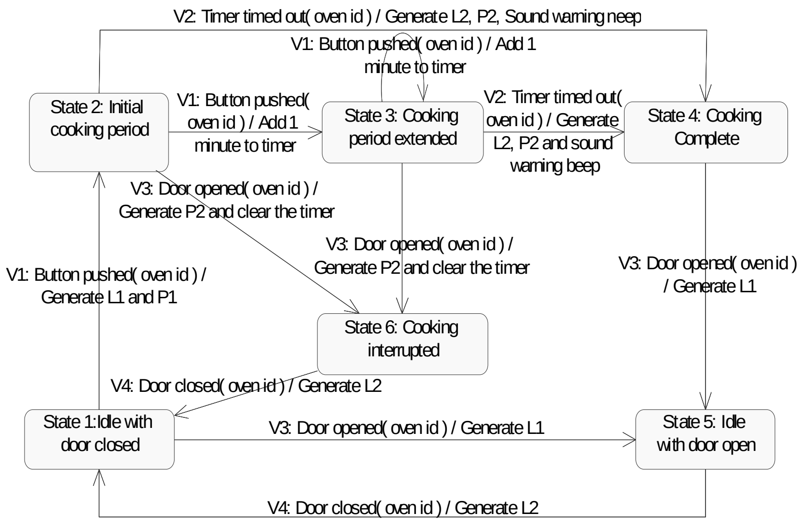
V2: Timer timed out(oven id) / Generate L2, P2, Sound warning neep
V1: Button pushed(oven id) / Add 1 minute to timer
State 2: Initial cooking period
V1: Button pushed(oven id) / Add 1 minute to timer
State 3: Cooking period extended
V2: Timer timed out(oven id) / Generate L2, P2 and sound warning beep
State 4: Cooking Complete
V3: Door opened(oven id) / Generate P2 and clear the timer
V3: Door opened(oven id) / Generate P2 and clear the timer
V3: Door opened(oven id) / Generate L1
V1: Button pushed(oven id) / Generate L1 and P1
State 6: Cooking interrupted
V4: Door closed(oven id) / Generate L2
State 1:Idle with door closed
V3: Door opened(oven id) / Generate L1
State 5: Idle with door open
V4: Door closed(oven id) / Generate L2

Fig. 9: Statechart Diagram of Oven

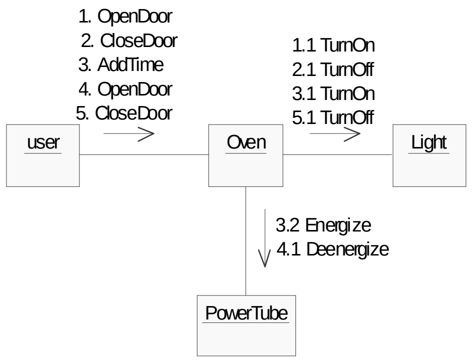

Fig. 10: Collaboration Diagram

By applying above rules, we obtain the following SPNM of microwave oven.

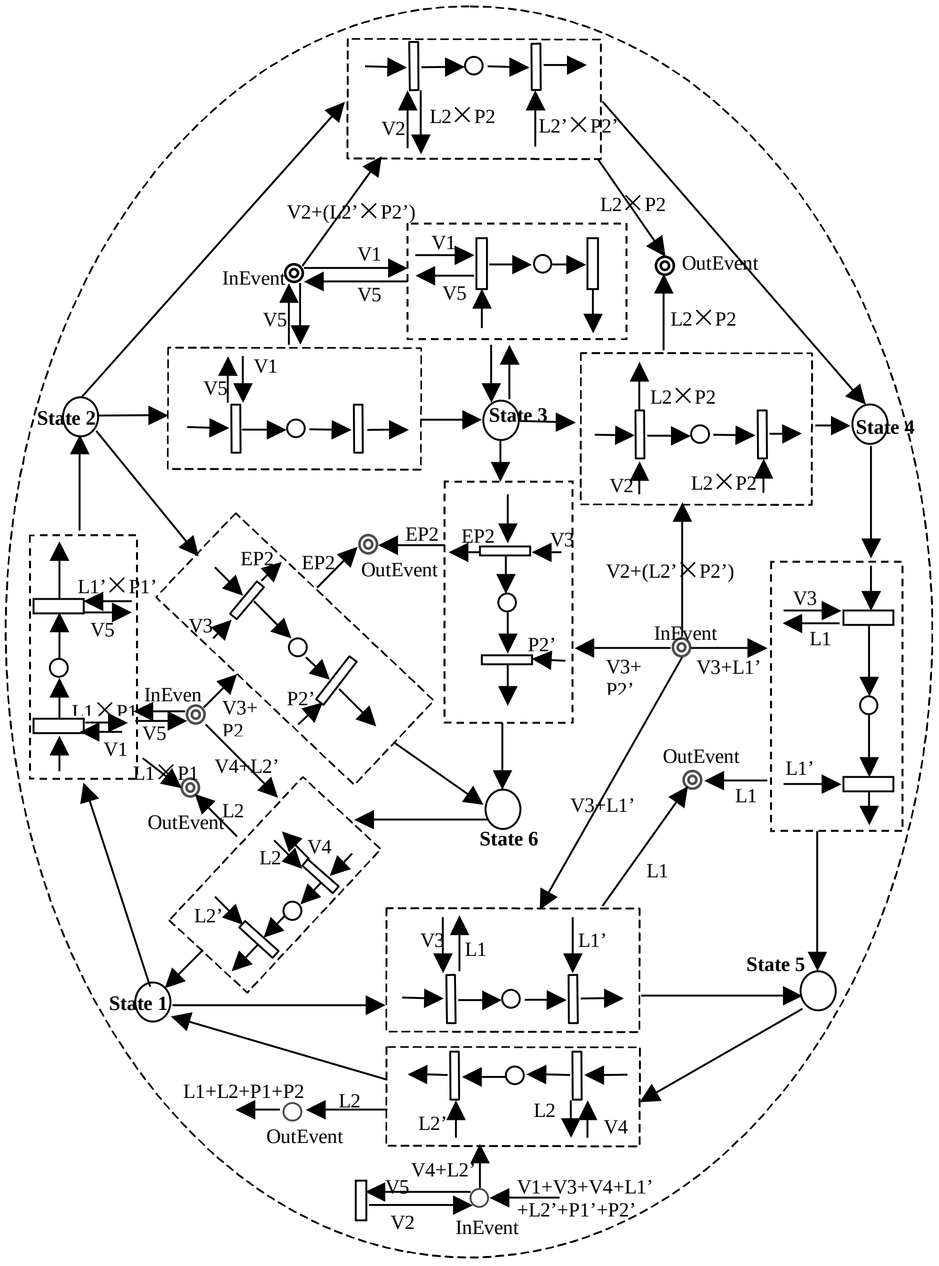

Fig. 11: The HPrTN Definition of Oven Statechart Diagram

From the Fig. 11, we know that every time the oven wants to notify the light/power tube to turn on or turn off, it must send a corresponding event to light/power tube, and wait for the corresponding event indicating the light/power tube has finished the action. When the door of the oven is closed and the user pushed button, an event V5 is generated, which means the oven need to change the cooking time. So “InEvent” place get the event V5 and transition “TimeMonitor” is enabled, when the transition is fired an event V2 is returned to ‘InEvent” place, which means the cooking was finished. For the actions that oven clear its timer and sound warning beep, it can be viewed as the actions of the corresponding “Leave” transition resulting in entering into state 4.

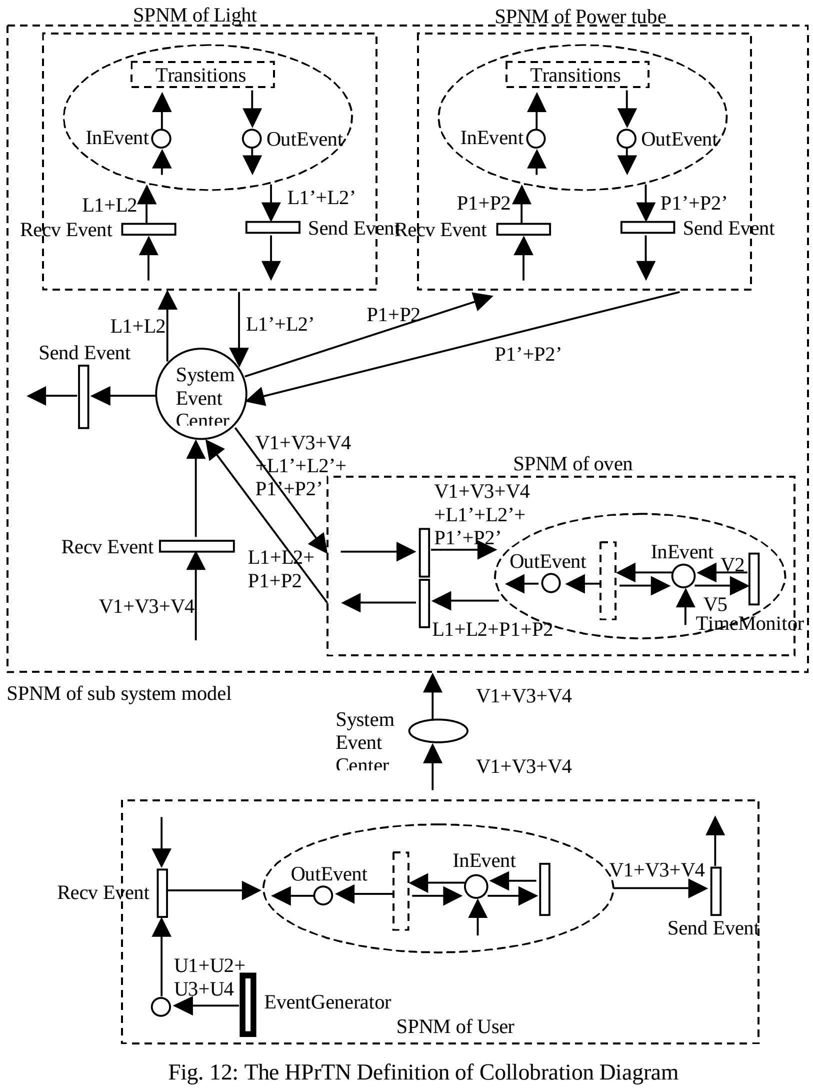

Fig. 12: The HPrTN Definition of Collobration Diagram

By adding “Recv Event”, “Send Event” transitions and the associated arcs, we get the SPNM of the subsystem model consisting of the SPNMs of the oven, the light and the power tube. The SPNM of subsystem model receives events from the “System Event Center”, which stores the events generated by the SPNM of the user. In the SPNM of subsystem model, “Send Event” transition is never fired because the subsystem model never sends events to the user.

By adding system inscription, we can simulate the executions of the system. The details are omitted here due to space limit.

6 Concluding Remarks

In this paper, we presented an approach to derive a SPNM in an HPrTN from a statechart diagram and to integrate SPNMs to obtain a complete system model according to the dynamic class relationships defined by collaboration diagrams. Each SPNM can be studied separately. By integrating SPNMs, we can detect potential inconsistencies among statechart diagrams of different classes. Since SPNMs are hierarchical, we can study the properties at a desired level of abstraction. SPNMs support both synchronous and asynchronous event communications.

One direction of future work is to generate Class Petri Net Models according to class diagrams and integrate them with SPNMs so that we can analyze structural and behavioral properties in a unified framework based on HPrTNs. Another direction of future work is to explore ways to formally analyze the properties of the system model using various Petri Net tools and ways to interpret the properties of system model with regard to original UML specifications that users can understand.

Acknowledgements

This work is support in part by NSF under grant HDR-9707076 and by NASA under grant NAG 2-1440.

Bibliography

[ADR01] G. Agha, F. De Cindio, and G. Rozenberg (eds.): Concurrent Object-Oriented Programming and Petri Nets – Advances in Petri Nets, Lecture Notes in Computer Science, vol.2001, Springer-Verlag, 2001.

[BPP99] R. Back, L. Petre, and I. Paltor, “Analyzing UML Use Cases as Contracts”, Proc. Of UML’99, Lecture Notes in Computer Science, vol. 1723, 1999, 518-533.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson: Unified Modeling Language User Guide, Addison-Wesley, 1999.

[Eva98] A. Evans, “Reasoning with UML Class Diagrams”, Proc. of 2nd IEEE Workshop on Industrial-Strength Formal Specification Techniques, Boca Raton, 1998, 102-113.

[FEL98] R. France, A. Evans, K. Lano, and B. Rumpe, “Developing the UML as a Formal Modeling Notation”, Computer Standards and Interfaces, no. 19, 1998, 325-334.

[HD01] X. He and Y. Ding, “Object-Orientation in Hierarchical Predicate Transition Nets”, Lecture Notes in Computer Science, vol2001, Spring-Verlag, 2001, 196-215.

[He96] X.He, “A formal definition of hierarchical predicate transition nets”, Proc. Of the 17th Int’l Conference on the Application and Theory of Petri Nets (Lecture Notes in Computer Science, vol. 1091, June, Osaka, Japan, 1996, 212-229.

[He00a] X. He, "Formalizing Class Diagrams Using Hierarchical Predicate Transition Nets", Proc. of the 24th Int'l Computer Software and Application Conference (COMPSAC' 2000), Taiwan, Oct. 2000, 217-222.

[He00b] X. He, "Defining UML Class Diagrams using Hierarchical Predicate Transition Nets," Proc. of the Workshop on Defining Precise UML Semantics in ECOOP'2000.

[He00c] X. He, "Formalizing Use Case Diagrams in Hierarchical Predicate Transition Nets", Proc. of the IFIP 16th World Computer Congress, Beijing, China, August 2000, 484-491.

[HL91] X. He and J.A.N. Lee: "A Methodology for Constructing Predicate Transition Net Specifications", Software – Practice & Experience, vol.21, no.8, 1991, 845-875.

[KC99] S. Kim and D. Carrington, "Formalizing the UML Class Diagram Using Object-Z," Proc. Of UML'99, Lecture Notes in Computer Science, Vol. 1723, 1999, 83-98.

[Kna99] A. Knapp, "A Formal Semantics for UML Interactions," Proc. Of UML'99, Lecture Notes in Computer Science, Vol. 1723, 1999, 116-130.

[Lak01] C. Lakos: "Object-Oriented Modeling using Object Petri Nets", Lecture Notes in Computer Science, vol.2001, 2001, 1-37.

[LB98] K. Lano and J. Bicarregui, "Formalizing the UML in Structured Temporal Theories," Proc. Of the 2nd ECOOP Workshop on Precise Behavioral Semantics, Springer-Verlag, 1998, 105-121.

[Mur89] T.Murata, "Petri Nets: Properties, Analysis and Applications", Proc. Of IEEE, vol. 77, no.4, 1989,

[OMG00] OMG Unified Modeling Language Specification, Version 1.3, 1st Edition, March, 2000.

[OP99] G. Overgaard and K. Palmkvist, "A Formal Approach to Use Cases and Their Relationships," Proc. of the Unified Modeling Language: UML'98: Beyond the Notation, Lecture Notes in Computer Science, Vol. 1618, Springer-Verlag, 1999.

[SF97] M. Shroff and R. France, "Towards a Formalization of UML Class Structures in Z," Proc. of COMPSAC'97, Washington, D.C., 1997.

[SS00] J. Saldhana and S. M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," Proc. of the Int. Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, July 2000, 103-110.

[SM92] Shlaer and Mellor, Object Lifecycles – Modeling the world in states, Yourdon Press, Prentice Hall, 1992.