Consistent Integration between Object Oriented and Coloured Petri Nets Models

Bassam Rajabi and Sai Peck Lee
Faculty of Computer Science and Information Technology, University of Malaya, Malaysia

Abstract: *Unified Modeling Language (UML) is easier to understand and communicate using graphical notations, but lacks techniques for model validation and verification especially if these diagrams are updated. Formal approaches like Coloured Petri Nets (CPNs) are based on strong mathematical notations and proofs as basis for executable modeling languages. Transforming UML diagrams to executable models that are ready for analysis is significant, and providing an automated technique that can transform these diagrams to a mathematical model such as CPNs avoids the redundancy of writing specifications. The use of UML diagrams in modeling Object Oriented Diagrams (OODs) leads to a large number of interdependent diagrams. It is necessary to preserve the diagrams consistency since they are updated continuously. This research proposes a new structure for the mutual integration between OODs and CPNs modeling languages to support model changes, the proposed integration suggest a new structure (Object Oriented Coloured Petri Nets (OOCPN)) to include set of rules to check and maintain the consistency and integrity of the OOCPN model based on OODs relations.*

Keywords: *CPNs, consistency rules, OODs, OOCPN, UML.*

Received September 13, 2012; accepted March 24, 2013; published online April 4, 2013

1. Introduction

Software modeling is one of the most important activities for large-scale software development. Object Oriented (OO) modeling language is widely used in software analysis and design. Unified Modeling Language (UML) as an OO modeling technique supports a variety of diagrams to model software systems from different perspectives using UML structural, behavioural, and interaction diagrams. UML diagrams are interrelated; some components for one diagram may be derived from other diagrams. Since UML diagrams can be divided into different categories, where each category focuses on a different perspective of a problem domain, one of the critical issues is to keep consistency among diagrams [12, 15, 26].

UML is powerful in describing the static and dynamic aspects of systems, but remains semi-formal and lacks techniques for diagrams validation and verification especially if these diagrams updated continuously. Formal specifications and mathematical foundations such as Coloured Petri Nets (CPNs) are used to automatically validate and verify the behaviour of the model. The main advantages expected from the integration of OO and CPNs modeling languages are: better representation of systems complexity as well as ease to adapt, correct, analyze, and reuse a model. This integration is based on the combination of the best characteristics of CPNs and OO design methods. UML has a powerful structuring capability to describe the static aspects of systems, and CPNs are powerful in modeling system dynamics and behavioural aspects [9, 13, 21, 27]. CPNs complement UML modeling in providing a powerful and visual formalism for specifying behaviour, especially the concurrent behaviour in executable notation. CPNs have a static part and a dynamic behaviour part. The static part describes the structure of the nets, and the dynamic behaviour part describes how the states can change during simulation [22].

Transforming UML diagrams to executable models that are ready for analysis is significant, and providing an automated technique that can transform these diagrams to a mathematical model such as CPNs avoids the redundancy of writing specifications. Many approaches for the integration of OO and CPNs have been investigated and developed. This research aims to consider the neglected aspects, particularly, the transformation of OODs (which are represented by the UML structural, behavioural, and interaction diagrams) to OOCPN model including consistency and integrity rules for supporting model changes. In addition, rules to maintain the consistency and integrity of OOCPN model during the transformation process and after updating OOCPN model elements are proposed based on OODs relations. The rest of the paper is organized as follows: In section 2, a brief overview on related works is provided. In sections 3 and 4, the transformation between the UML structural, behavioural, and interaction diagrams to OOCPN is proposed based on the transformation rules, which include rules to check diagrams consistency and integrity. Transformation rules modeling and validation are provided in section 5. Finally, we concludes the research and presents the future work in section 6.

2. Related Works

UML is one of the most widely used for OO modeling. It provides a variety of diagrams with powerful syntax and semantics to build precise models for software development [26]. UML supports variety of diagrams to model software systems from structural, behavioural, and interaction perspectives.

Most graph-based modeling languages have their root in Petri Net (PN) theory. Places and transitions are the main components of PN model, and arcs are used to connect between them. The main characteristics of the Coloured Petri Net (CPN) are data structures and hierarchical structure. These characteristics are used to represent the object dynamics and to check the models correctness [1, 13]. Object PNs extend the formalism of CPNs with OO features, including inheritance, polymorphism, and dynamic binding [18]. Timed Petri Nets (PNs) introduce time in PNs. A framework to transform UML state chart and collaboration diagrams to CPNs is proposed in [11] to provide a dynamic model analysis. State chart diagrams are converted to CPNs, and collaboration diagrams are used to connect the state charts into a single CPN model. Object PN Models (OPMs) [24] are used to generate a PN model from UML object diagram. Object classes and states classes are represented using CPN places, while object instances are represented by CPN tokens. Generating object CPNs from UML state chart diagrams was proposed in [2]. An abstract node approach is used to transform an OO model into a hierarchical CPN model [1]. Using this approach, class and sequence diagrams can be transformed to CPNs. Transforming UML 2.0 sequence diagrams into CPNs is presented in [8, 14].

The approach by Shin *et al.* [25] is to model the transformation of UML use case, class, and collaboration diagrams to CPN models. Integrating OO design with CPN was developed by [19] for analysis purpose. The CPN model is used to verify the UML diagrams before the implementation. Meta-modeling and formalism transformation framework is a general framework for the analysis of software systems using model-checking [10]. This framework transforms the UML model into PNs for further analysis. The UML model is composed of classes, state charts, and sequence diagrams. A Hierarchical Object Oriented Petri Net integrates HPN with OO concepts to support OO features including abstraction, encapsulation, modularization, message passing, inheritance, and polymorphism [28]. A meta-level and highly automated technique based on graph transformation approach is presented in [30]. This approach formally transforms UML state charts and behavioural diagrams to PNs for verification. A methodology to derive CPNs from UML object, sequence, state chart, and collaboration diagrams is proposed in [3, 4]. Some of the PN modeling languages adapt the OO concepts in PN called Object Oriented Petri Nets (OOPN) as in [20]. The main concepts of these approaches are: OOPN is a set of class nets; a class is specified by a set of object nets, method nets, synchronous ports, negative predicates, and message selectors; object nets and method nets can be inherited; a token represents an object or instance of class; synchronous ports are special transitions which cannot fire alone but only dynamically fused to some regular transitions. A comparative study for software tools that support the transformation of UML static and dynamic diagrams to PNs/ CPNs models is provided in [21]. Some of these tools are ArgoSPE tool and WebSPN. This study indicates that the transformation approaches have certain weaknesses, such that, each transformation approach uses only a subset of the UML diagrams, and most of these transformations are based on the behavioural UML diagrams. A comparative study between approaches that transform the UML diagrams to CPNs is discussed in [21].

3. Proposed OODs to OOCPN Transformation Rules

Many approaches for integrating OO modeling and PNs/CPNs have been investigated and developed. The transformation between UML diagrams and CPNs is partially supported for a subset of the UML diagrams. This research focuses on the transformation of UML diagrams from the structural, behavioural, and interaction perspectives. This includes rules to maintain the consistency and integrity of the OOCPN model to support model changes. The block diagram of the transformation process is shown in Figure 1.

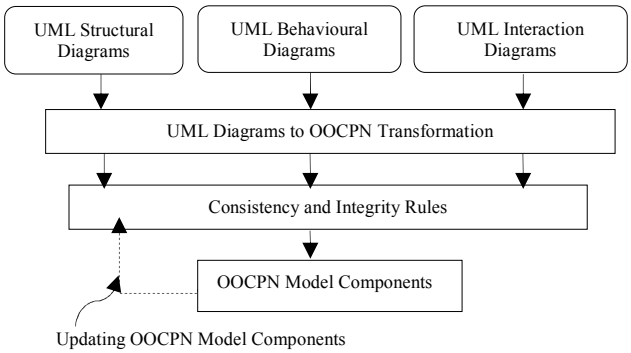

Figure 1. Block diagram for transformating UML diagrams to OOCPN model.

The components of UML structural, behavioural, and interaction diagrams are transformed to CPNs elements based on the proposed transformation rules. Consistency and integrity rules are checked during the transformation process and after updating the CPNs model. We apply the proposed OOCPN structure on four UML diagrams (class diagram, object diagram, activity diagram, and sequence diagram). These diagrams cover three perspectives of UML diagrams (structural, behavioural, and interaction perspectives). Activity and sequence diagrams implement the same

thing but activity diagram concerns on the control flow and sequence diagram concerns on the object flow so the coevolution between these diagrams is very high.

The general structure for the OOCPN model after the transformation of UML diagrams will be as follows: Attributes and operations are transformed from the class diagram. These attributes and operations are used by other OOCPN model components. Classes are organized into subpages or subnets. These subpages can be instantiated using tokens, which represent the objects. Objects behaviour and interaction are described using the transformed behavioural and interaction diagrams. Activity diagram describes the flow of control form activity to activity. Sequence diagram describes the flow of control from object to object. Sequence diagram focuses on the times that messages are sent. The proposed structure can be described formally as a tuple of:

<OOCPN structure, Relations, Rules>

OOCPN structure is described formally as in Definition 1. OOCPN model elements are grouped together according to UML diagrams relations. Rules are used to maintain the consistency and integrity of the transformed model as discussed in Definition 2.

- *Definition 1: Proposed OOCPN Structure*

The Proposed OOCPN structure is defined by the tuple:

$$n = (\Sigma, Pg, P, Fp, T, SubT, A, N, C, G, E, M_0, R)$$

Where:

Σ: Is a finite set of non-empty types, called colour sets.

Pg: $\{pg_0, ..., pg_n\}$ is a set of pages, where pg_0 is the main page.

P: $\{p_1, p_2, \ldots, p_n\}$ is a finite set of places.

Fp: $\{fp_1, fp_2, \ldots, fp_n\}$ is a finite set of fusion places.

T: $\{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions.

SubT = $\{Subt_1, \ldots, Subt_n\}$ is a finite set of substitution transitions.

A: Represents a set of directed arcs.

N: Is a node function.

C: $P \rightarrow \Sigma$ is a colour function.

G: Is a guard function.

E: Is an arc expression function.

M_0: $P \rightarrow C$ is the initial (coloured) marking.

R: $\{r_1, \ldots, r_n\}$ is a finite set of consistency and integrity rules.

- *Definition 2: OOCPNs Model Relations and Rules*

The Proposed transformation rules are used to transform the UML diagrams' elements to OOCPN elements. OOCPN elements are grouped together according to the UML diagrams' relations as follows: Let O be an OO software system represented by a set of UML diagrams elements (E), where $E_o = \{E_1, E_2, \ldots, E_n\}$. Let $TR_o = \{TR_1, TR_2, \ldots, TR_n\}$ be the set of transformation rules. Let $OOCPN_o = \{OOCPN_1, OOCPN_2, \ldots, OOCPN_n\}$ be the set of equivalent OOCPN elements of E_o. We can define the transformation rule between $\{E_j, OOCPN_j\}$ as follows: $\forall$ Diagram element $\in$ E: $E_j \Rightarrow$OOCPNj// where j is a specific diagram element. OODs are organized in OOCPNs as a set of {S, B, and I}. Where *S*: Is for the structural diagrams' elements, *B*: Is for the Behavioral diagrams' elements, and *I*: Is for the Interaction diagrams' elements. OODs elements in OOCPNs are a set of:

{S (E_1, E_2, ..., E_n), B (E_1, E_2, ..., E_n), I (E_1, E_2, ..., E_n)}
{CD (E_1, ..., E_n), OD (E_1, ..., E_n), AD (E_1, ..., E_n), SD (E_1, ..., E_n)}

Where, *CD:* Class Diagram, *OD:* Object Diagram, *AD:* Activity Diagram, and *SD:* Sequence Diagram. The proposed transformation rules include information about the following: Rules to transform UML diagrams' elements to OOCPN as discussed in Definition 2. Consistency and integrity rule (s) to maintain the consistency and integrity during the transformation and after updating OOCPN model components. These rules have the structure:

If (set of input conditions) then (set of output conditions) Else (set of output conditions)

This research proposes two rules (rules 1 and rule 2) to maintain the integrity after updating OOCPN model elements, and rule 3 is a set of consistency constraints to be checked during the transformation process:

- *Rule 1: Deleting a Referenced Element*

If (an update is to delete a referenced element) then (deleting the referenced element is not allowed)

Where a referenced element is an element defined by another diagram. For example, all diagrams' attributes must be defined in the CD. The following update operations are determined based on rule 1, delete a CD attribute, operation, class, class inheritance, association, or navigability arrow. Delete an object in the OD, SD:

- *Rule 2: Modifying a Referenced Element*

If (an update is to modify a referenced element) then (modifying the referenced element is not allowed)

The following update operations are determined based on rule 2, modify a CD attribute name, operation name, class name, inherited class name, navigability arrow direction, polymorphic operation name, or interface element name. Modify an object name in the OD, SD. Modify a SD message name or a message attribute name.

- *Rule 3: Consistency Constraints*
  - *Rule 3.1:* The class attribute name and the association role name cannot have the same name [5].
  - *Rule 3.2:* Two associations with the same name and role name are not allowed.

- *Rule 3.3:* No private/ protected attribute or operation can be accessed by an operation of another class.
- *Rule 3.4:* All diagrams' attributes/ operations must be defined in the CD.
- *Rule 3.5:* A cycle is not allowed in the directed paths of aggregation links.

4. Transformation of UML Diagrams Elements to OOCPN

A class diagram is useful to represent information of actors, roles, organizational unit, and relevant data. Actors and data stores are objects in the object diagram. Processes are implemented as operations in the object model. A process may be split over a number of operations [17, 29]. Both sequence and activity diagrams describe the dynamic behavior of use cases. An activity diagram is used to model the business logical steps and dynamic behavior derived from the use cases [6]. It concentrates on the dynamical relations among business activities [29]. Sequence diagram is used to describe the interactions among business objects based on messages. The sequence diagram focuses on the messages' times. In this section, transformation rules for transforming UML class, object, activity, and sequence diagrams' elements to OOCPN are provided.

4.1. Class Diagram Transformation Rules

Class diagram elements are transformed to OOCPN according to the following transformation rules:

CD attribute ⇒ CPN place
Consistency and integrity rule:
If (a new attribute is created) then ((the attributes name should be unique in the class) ⇒ (Rule 3.1 is verified))
CD attributes type ⇒ CPN colour set
CD values ⇒ CPN tokens
CD value type ⇒ CPN colour set
CD operation ⇒ CPN subpage
Consistency and integrity rules:
If (a new operation is created) then (Rule 3.3 is verified)
If (an operation has a pre or post condition attribute) then ((Rule 3.4 is verified) ⇒ (attribute type must be compatible))
CD class transformation to CPNs
CD class ⇒ CPN subpage
CD class instance ⇒ CPN substitution transition
CD class name and attribute ⇒ CPN place with appropriate colour type
Consistency and integrity rule:
If (the created class contains at least one abstract operation) then ((it must be declared abstract) ∩ (it cannot be instantiated or invoked))
CD communication method and dynamic binding transformation to CPNs
CD synchronous request ⇒ CPN transition fusion
CD asynchronous request ⇒ CPN fusion places
Consistency and integrity rules are the same as in the SD message transformation to CPNs.
SD synchronous and asynchronous messages are transformed to CPNs the same as in CD communication methods and dynamic binding:
CD generalization ⇒ Hierarchical Coloured Petri Net (HCPN) by net addition (place and/or transition fusion)
Consistency and integrity rule:
If (a class element is part of an inheritance relation ∩ it is not a polymorphic operation) then
(All sub classes are not allowed to redefine the inherited element)
(classes that have the "leaf" property cannot be extended)
(classes that have the "root" property cannot extend another classes)
CD associations ⇒ CPN places connected between the classes' subnets
Consistency and integrity rule:
If (a new association is created/ modified) then
(Rule 3.1 and Rule 3.2 are verified)
CD aggregation ⇒ HCPN by net addition (place and/ or transition fusion)
Consistency and integrity rule:
If (a new association is created/ modified) then
(Rule 3.1, Rule 3.2, and Rule 3.5 are verified)

The aggregation relation means that the target subnet needs to contain some instances of the source subnet. Communication between subnets will be the same as in the CD communication methods and dynamic binding. Composition (is-part-of) can be modeled the same as in aggregation, but the difference is that the target subnet needs to contain one instance of the source subnet.

CD navigability arrow ⇒ CPN arc
Consistency and integrity rules:
If (a navigability arrow is created) then
If (an association type is generalization) then
(the navigability arrow should be from the sub class to the super class)
Else if (an association type is aggregation) then
(Rule 3.5 is verified)
CD polymorphism ⇒ HCPN by net addition (place and/or transition fusion)
Consistency and integrity rule:
If (a polymorphism operation is created) then
((Rule 3.4 is verified) ∩ (It can override by the subclasses of its class))
CD multiplicity ⇒ CPN tokens and substitution transition
Consistency and integrity rules:
If (a multiplicity range is created) then
(the multiplicity range for an attribute must be adhered to by all elements that access it)
(Class's multiplicity must not be violated by the multiplicity of any association end in which it is the participant)
Else if (a multiplicity range is modified) then
(it must be consistent/ correct after the modification)
CD role name ⇒ CPN auxiliary text
Consistency and integrity rule:
If (a role name is created or modified) then

(Rule 3.1 and Rule 3.2 are verified)

CD interface ⇒ the same as in CD class transformation except that it lacks instance variables and implemented methods [7]

Consistency and integrity rule:
If (an interface element/ operation is created)
then (Rule 3.4 is verified)

4.2. Object Diagram Transformation Rules

Object diagram elements are transformed to OOCPN according to the following transformation rules:

OD object transformation to CPNs

OD and SD object (class instance) ⇒ CPN tokens

Number of tokens is equal to ($\sum Occ_i$, $i > 0$. where Occ_i is the number of instances).

OD object attribute ⇒ CPN token colour

Consistency and integrity rule:
If (a new object is created) then ((an object name should be unique) ⇒ (it represents an instance of a class in the CD))

OD object states transformation to CPNs

OD instance variable ⇒ CPN place

OD variable type ⇒ CPN place colour

OD message data type ⇒ CPN product data type supported in CPNs for all the message attributes

OD behavior transformation to CPNs is the same as in the CD operation transformation

OD communication transformation to CPNs is the same as in SD messages transformation

Consistency and integrity rule:
If (a new type is created/ modified) then (it must be defined in the CD attributes or operations types)

4.3. Activity Diagram Transformation Rules

Activity diagram elements are transformed to OOCPN according to the following transformation rules:

AD sub-activity ⇒ CPN subpage

Consistency and integrity rules:
If (a sub-activity is created) then (It should represent an operation in the CD)
If (an activity has a pre or post condition attribute) then (Rule 3.4 is verified)

AD action ⇒ CPN transition (it takes a specific input from some places and produces a specific output to places)

Consistency and integrity rule:
If (an action has a pre or post condition attribute) then (Rule 3.4 is verified)

AD control flow ⇒ CPN places with input/ output arcs

Consistency and integrity rule:
If (a control flow is created) then (two AD elements are connected using the created control flow)

AD object flow transformation to CPNs

AD object flow ⇒ CPN places with input/output arcs

AD object node ⇒ CPN place

AD control nodes (Fork, Join, and Merge) transformation to CPNs

AD control node ⇒ CPN transition

AD control node input and output flow ⇒ CPN places

AD decision node ⇒ CPN arc inscription

AD activity sequence transformation to CPNs

AD activity sequence ⇒ CPN page including a set of interconnected activities

AD activity ⇒ CPN transition

AD activity input and output ⇒ CPN places

Consistency and integrity rule:
If (a new sequence of activities is created/deleted/ modified) then (∃ the same sequence of activities in the AD)

AD start/ end state transformation to CPNs

AD start node ⇒ CPN place without any incoming arc

AD end node ⇒ CPN place without any outgoing arc

Consistency and integrity rule:
If (a start or end node is created/ deleted) then (one start/end node only is founded in the AD)

4.4. Sequence Diagram Transformation Rules

Sequence diagram elements are transformed to OOCPN according to the following transformation rules:

SD message ⇒ CPN transition

Consistency and integrity rules:
If (a message is created) then begin
If (a message is to call an operation) then ((Rule 3.4 is verified) ⇒ (the object in the SD can only invoke an operation on another object if it has a navigable association to it in the CD))
If (a message name and an object name are the same) then (the message name is not allowed to be the same as the object's name)
If (a message is to call a private/protected operation) then (Rule 3.3 is verified)
If (a message is to invoke an abstract operation) then (the abstract operation cannot be invoked in the SD)
If (an operation that is called by the message has a pre or post condition attribute) then (Rule 3.4 is verified)
If (attributes are assigned to return messages/ operations values) then (attributes types have to be compatible)
End if

Transforming the following diagrams' elements to CPNs are the same as message transformation to CPNs: AD Call Behavior, SD operation call, SD creation and deletion:

SD and AD condition ⇒ CPN place

SD action bars/ lifelines ⇒ CPN places to represent the beginning and the end of the action bar [26]

Consistency and integrity rule:
If (an action bar/ life line is created/ modified) then (it should represents sequence of activities in the AD)

5. Transformation Rules Modeling and Validation

In this section, transformation rules modeling and validation are provided with examples. CPN Tools version 3.4 [27] is used to model and simulate the proposed OOCPN structure. The following Figures: 2 to 18 are examples with some description of the

transformation for the basic diagrams elements to OOCPN based on the proposed structure where transformation and consistency rules are applicable in any system modeled using CPNs diagrams.

CD operation can be modeled as subpage in CPNs. Substitution transition is used to call the operation as shown in Figure 4. Substitution transitions (Method 1 and Method 2) can be recognized by double boxes and subpage tags positioned next to them, tags (Method1 and Method 2) contains the name of the related subpage. Operation parameters are: pre and post conditions, and messages to be transferred. "[ClassName A or B]" is an example for the operation precondition. Sometimes the transition components (input, output, guard, and action) are enough to model the operation. CD classes with their data fields and functions are modeled as subnets in CPNs. These subnets can be instantiated many times using substitution transitions. For example, the class diagram in Figure 2 is transformed to CPNs as in the following CPN code:

var Operation_Date: STRING;
var Amount: STRING;
*colset Transaction = product STRING * STRING;*
//Transaction class colour set is a product of the class attributes' colours

Operations are transformed to subpages. Subpages are only accessed by the class name "[ClassName A or B]" as shown in Figure 4. Abstract classes are modeled in CPNs the same as in classes, but abstract classes have no instances.

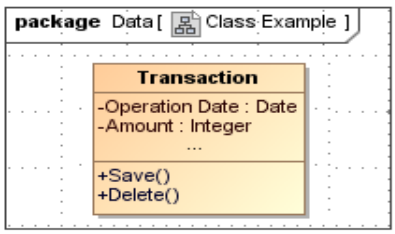

Figure 2. An example class diagram.

CD Communication methods and dynamic binding are transformed to CPN transition and place fusion. A substitution transition is an example of transition fusion as shown in Figure 4 (Method 1 and Method 2 tags). Figure 3 is an example for fusion places. A synchronous request is modeled by a transition fusion, such that the CPN model cannot continue its own process until the reply token is arrived from the substitution transition subpage. An asynchronous request is modeled by fusion places. Place fusion is a method to transfer a token to all fusion places at the same time. Fusion places can be used for concurrent processing [13, 18]. CD generalization/ class inheritance and dynamic of objects can be represented using HCPN by net addition (place and/ or transition fusion). Modularity is also supported by HCPN. Figure 4 shows the transformation of generalization to CPNs.

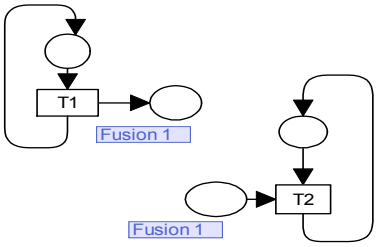

Figure 3. Place fusion example.

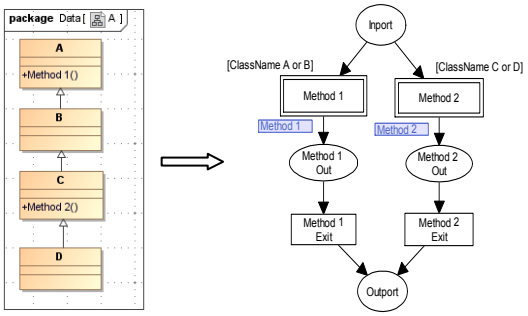

Figure 4. CPNs for generalization/inheritance.

CD Associations are transformed to CPN places connected between the classes' subnets. These subnets communicate between themselves using the transition and place fusions as shown in Figures 3 and 4. CD polymorphism can be transformed to CPNs through net structure (HCPN), in addition to the net inscription as shown in Figure 5.

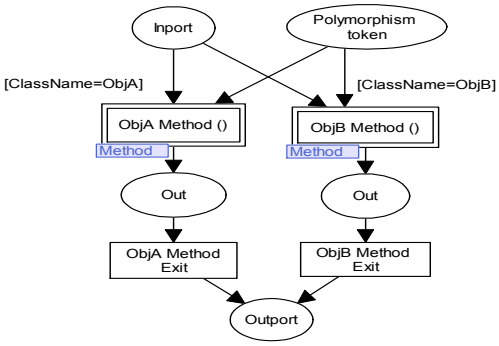

Figure 5. CPNs for polymorphism.

An inherited attribute (polymorphism token) can hold tokens of the super and subclasses. It is connected to the transition which represents the overriding operation. OD Object (class instance) and attribute transformation is shown in Figure 6 (object "C1: Class" is transformed to CPN place). AD control nodes (Fork, Join, and Merge) are modeled as a CPN transition. Each input flow and each output flow of the control node is modeled by a CPN place as shown in Figures 7 and 8. The merge node and the decision node have the same notation, but in the merge node there are multiple inputs and one output [16].

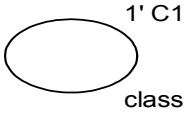

Figure 6. Object instance modeled in CPNs.

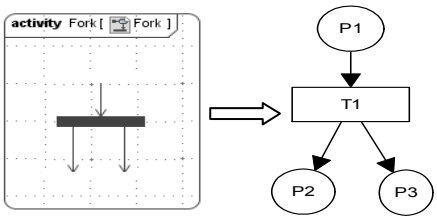

Figure 7. Transformation of fork node to CPNs.

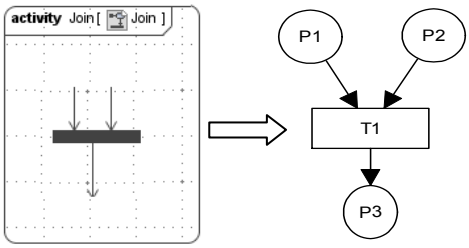

Figure 8. Transformation of join node to CPNs.

AD Decision node is represented in CPNs by arc inscription to control the tokens passing. Tokens represent the variables' values. Each activity connected to the transition node is transformed to a CPN transition as shown in Figure 9. AD branch has the same transformation such that each decision node represents a branch.

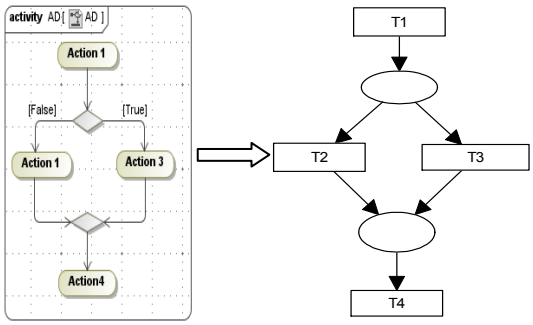

Figure 9. Transformation of decision node to CPNs.

AD start/ end node and activity sequence transformation to CPNs are shown in Figure 10.

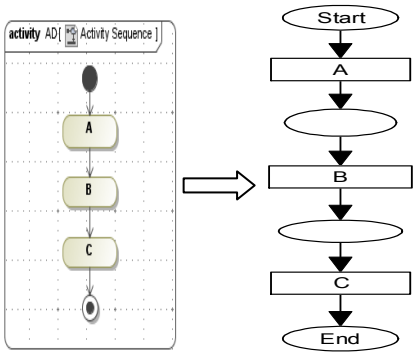

Figure 10. Transformation of activity sequence to CPNs.

AD and SD Activity Iteration/ Loop transformation to CPNs is shown in Figures 11 and 12. SD messages are transformed to CPN transitions as shown in Figure 13. The order of transitions is according to the order of the messages in the sequence diagram. Tokens flow between places and transitions are modeled to fire the transitions (execution of messages). Places represent the objects used during the messages' execution. A SD and AD condition is transformed into a CPN place. An example for the condition is provided in Figure 13 ([a == True]), and it is transformed to CPNs as shown in Figure 14. It is represented as a constraint to fire the transition.

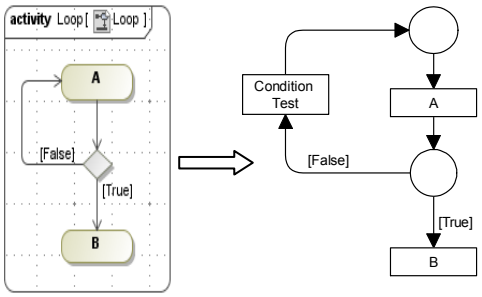

Figure 11. Transformation of iteration/ loop to CPNs.

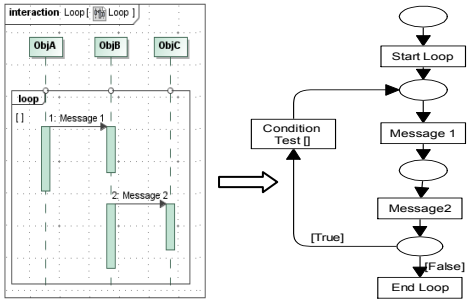

Figure 12. Transformation of sequence diagram loop to CPNs.

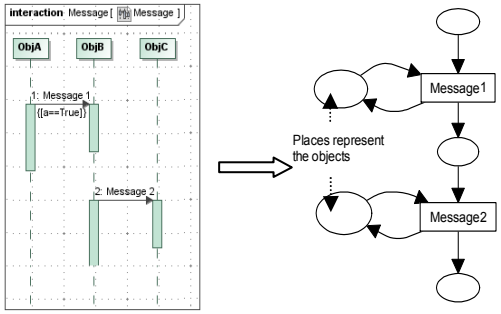

Figure 13. Transformation of sequence diagram messages to CPNs.

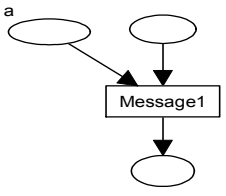

Figure 14. Sequence diagram conditions modeled in CPNs.

alt (alternative choice) is to represent choices (nested branches). Each choice is transformed to CPNs as in the messages transformation. Choices are selected for execution based on the true value of the choice guard. The branches are combined together using shared input and output places as shown in Figure 15.

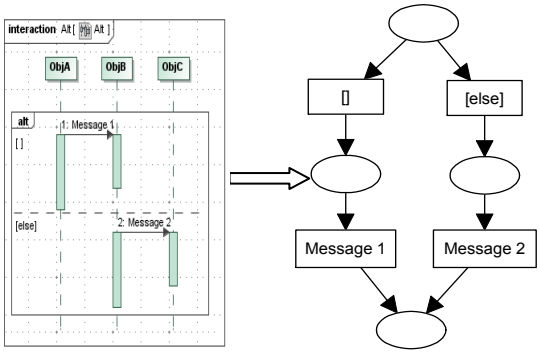

Figure 15. Transformation of alt operator to CPNs.

opt (optional operator) can be transformed to CPNs the same as in the alt operator, because opt is considered as an alternative choice with only one branch whose guard is not the else [23]. ref construct is transformed to a CPN substitution transition to include/reuse a sequence diagram inside another sequence diagram. par (short for parallel) is used to represent number of branches occurred in parallel. Each branch is transformed to CPNs as in the messages transformation, then these branches are combined together using shared input and output places and transitions as shown in Figure 16. Figures 17 and 18 show a complete example for modeling adding a new SD massage and its consistency rules.

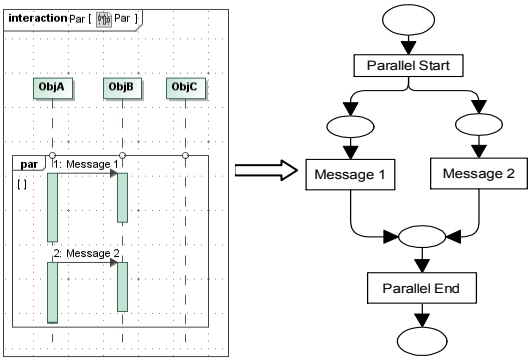

Figure 16. Transformation of par operator to CPNs.

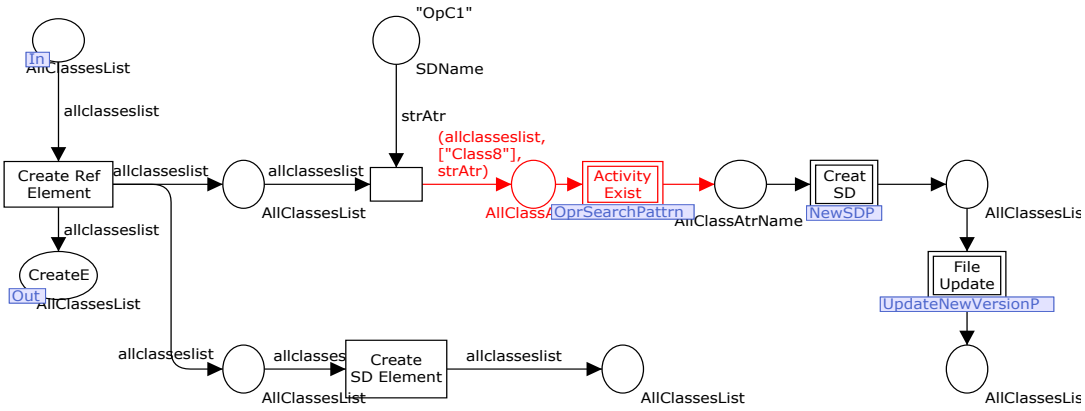

Figure 17. Adding new SD massages.

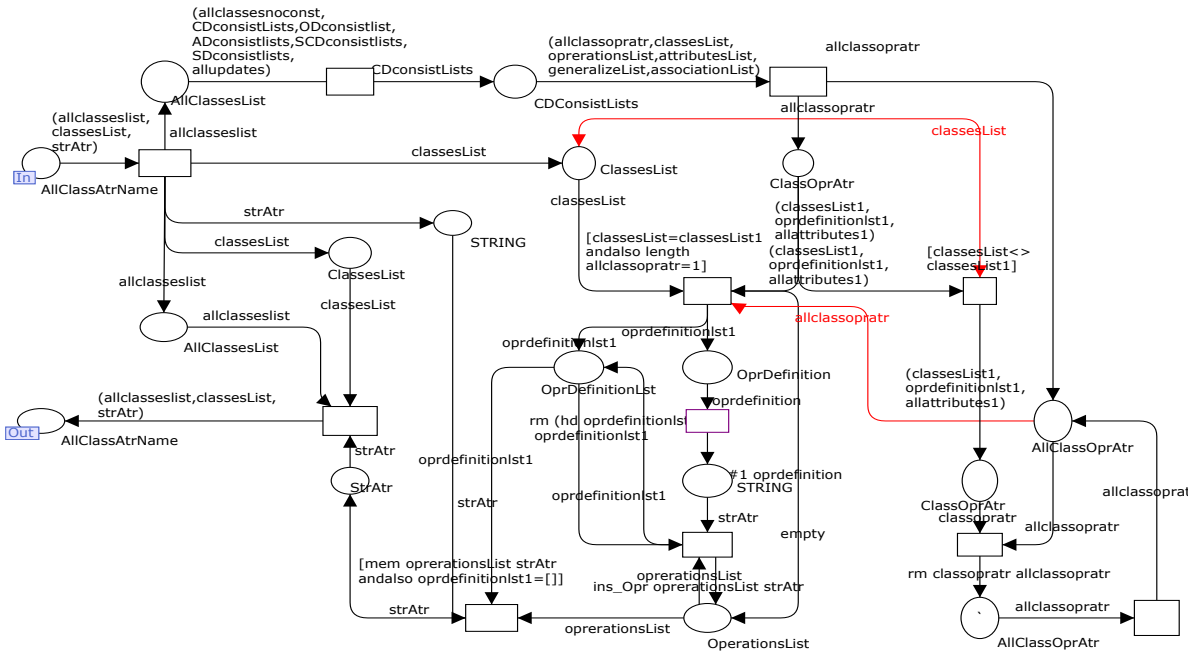

Figure 18. Adding new SD massegae consistency rule.

6. Conclusions and Future Work

The importance of software models in system design is growing. Recently, large-scale software projects focus on modeling more than programming. OO modeling language and PN modeling are most widely used in software analysis and design. This research combines the advantages of the formal and semi-formal modeling languages. The UML diagrams as semi-formal modeling language were used to provide powerful structuring capabilities in the model design. The CPNs as a formal and executable modeling language described the behaviour of UML model formally. Transformation rules are proposed to transform the OODs diagrams to OOCPN model. In addition, rules to maintain the consistency and integrity of the OOCPN model are proposed to support model changes. Consistency and integrity rules are based the OODs diagrams' relations and the proposed OOCPN structure. The future work of this research is to provide a change impact and traceability analysis techniques that automatically determine the impact of change in diagrams elements.

References

[1] Bauskar E. and Mikolajczak B., "Abstract Node Method for Integration of Object Oriented Design with Colored Petri Nets," *in Proceedings of the 3rd International Conference on Information Technology: New Generations*, Las Vegas, USA, pp. 680-687, 2006.

[2] Bokhari A. and Poehlman S., "Translation of UML Models to Object Coloured Petri Nets with a View to Analysis," *in Proceedings of Software Engineering and Knowledge Engineering*, San Francisco, USA, pp. 568-571. 2006.

[3] Bouabana-Tebibel T. and Belmesk M., "Formalization of UML Object Dynamics and Behavior," *in Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 4971-4976, 2004.

[4] Bouabana-Tebibel T. and Belmesk M., "Integration of the Association Ends within UML State Diagrams," *the International Arab Journal of Information Technology*, vol. 5, no. 1, pp. 7-15, 2008.

[5] Briand C., Labiche Y., and O'Sullivan L., "Impact Analysis and Change Management of UML Models," *in Proceedings of the International Conference on Software Maintenance*, Washington, USA, pp. 256-265, 2003.

[6] Chang L., Chen S., and Chen C., "Workflow Process Definition and their Applications in E-Commerce," *in Proceedings of the International Conference on Microelectronic Systems Education*, Washington, USA, pp. 193-200, 2000.

[7] Chung L., "Object-Oriented Analysis and Design," available at: http://www.utdallas.edu/ ~chung/Fujitsu/index.htm, last visited 2010.

[8] Fernandes J., Tjell S., Jorgensen J., and Ribeiro Ó., "Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net," *in Proceedings of the 6th International Workshop on Scenarios and State Machines*, Washington, USA, pp. 1-10, 2007.

[9] GroupTGI , "Welcome to the Petri Nets World," available at: http://www.informatik.uni-hamburg.de/TGI/PetriNets/, last visited 2010.

[10] Guerra E. and de-Lara J., "A Framework for the Verification of UML Models. Examples Using Petri Nets," *in Proceedings of VIII Jornadas Ingeniería del Software y Bases de Datos*, Alicante, Spain, pp. 325-334, 2003.

[11] Hu Z. and Shatz M., "Mapping UML Diagrams to a Petri Net Notation for System Simulation," *in Proceedings of International Conference on Software Engineering and Knowledge Engineering*, pp. 213-219, 2004.

[12] Ibrahim S., Idris B., Munro M., and Deraman A., "Integrating Software Traceability for Change Impact Analysis," *the International Arab Journal of Information Technology*, vol. 2, no. 4, pp. 301-308, 2005.

[13] Jensen K., Kristensen M., and Wells L., "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems," *the International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3, pp. 213-254, 2007.

[14] Khadka B., "Transformation of Live Sequence Charts to Colored Petri Nets," *Masters Project Report*, University of Massachusetts Dartmouth, USA, 2007.

[15] Lucas F., Molina F., and Toval A., "A Systematic Review of UML Model Consistency Management," *Information and Software Technology*, vol. 51, no. 12, pp. 1631-1645, 2009.

[16] Maqbool S., "Transformation of A Core Scenario Model and Activity Diagrams Into Petri Nets," *Master Thesis*, School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[17] Miller R., "Practical UML: A Hands-On Introduction for Developers," available at: http://bdn.borland.com/ article/ 0,1410,31863,00. html, last visited 2003.

[18] Miyamoto T. and Kumagai S., "Application of Object-Oriented Petri Nets to Industrial Electronics," *in Proceedings of the 33rd Annual Conference of the IEEE Industrial Electronics Society*, Taipei, Taiwan, pp. 64-69, 2007.

[19] Motameni H., Movaghar A., Shiraz B., Aminzadeh B., and Samadi H., "Analysis

Software with an Object-Oriented Petri Net Model," *World Applied Sciences Journal*, vol. 3, no. 4, pp. 565-576, 2008.

[20] Niul J., Zou J., and Ren A., "OOPN: An Object-Oriented Petri Nets and its Integrated Development Environment," *in Proceedings of IASTED International Conference on Software Engineering and Applications*, 2003.

[21] Rajabi B. and Lee S., "A Study of the Software Tools Capabilities in Translating UML Models to PN Models," *the International Journal of Intelligent Information Technology Application*, vol. 2, no. 5, pp. 224-228, 2009.

[22] Rajabi B. and Lee S., "Modeling and Analysis of Change Management in Dynamic Business Process," *the International Journal of Computer and Electrical Engineering*, vol. 2, no. 1, pp. 199-206, 2010.

[23] Ribeiro R. and Fernandez J., "Some Rules to Transform Sequence Diagrams into Coloured Petri Nets," *in Proceedings of the 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pp. 237-257, 2006.

[24] Saldhana J. and Shatz S., "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," *in Proceedings of International Conference on Software Engineering and Knowledge Engineering*, pp. 103-110, 2000.

[25] Shin E., Levis A., and Wagenhals W., "Transformation of UML-based System Model to Design/ CPN Model for Validating System Behavior," *in Proceedings of the 6th International Conference on the UML/Workshop on Compositional Verification of the UML Models*, San Francisco, USA, pp. 1-19, 2003.

[26] Shinkawa, Y., "Inter-Model Consistency in UML Based on CPN Formalism," *in Proceedings of the 13th Asia Pacific Software Engineering Conference*, Washington, USA, pp. 411-418, 2006.

[27] Westergaard M. and Verbeek H., "CPN Tools," available at: http://cpntools.org/, last visited 2012.

[28] Xiaoning F., Zhuo W., and Guisheng Y., "Hierarchical Object-Oriented Petri Net Modeling Method Based on Ontology," *in Proceedings of International Conference on Internet Computing in Science and Engineering*, Harbin, China, pp. 553-556, 2008.

[29] Yang J. and Chen C., "An Integrated Approach for Workflow Process Modeling and Analysis Using UML and Petri Nets," *MIS Review*, vol. 11, pp. 47-75, 2003.

[30] Zhao Y., Fan Y., Bai X., Wang Y., Cai H., and Ding W., "Towards Formal Verification of UML Diagrams Based on Graph Transformation," *in Proceedings of IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, Beijing, China, pp. 180-187, 2004.

Bassam Rajabi received his MSc degree in computer science from Alquds University, Jerusalem, Palestine, in 2005. Currently, he is a PhD student in computer science at University of Malaya, Malaysia. From 2001 to 2004, he was a research and teaching assistant with the Computer Science Department, Alquds University-Palestine. From 2001 to 2005 he was a lecturer with the Computer Science Department, ORT College-Palestine. He was a lecturer and Dean Assistant for Administrative Affairs from 2005 till Now with Wajdi University College of Technology-Palestine. His areas of interest are software design and modeling techniques.

Sai Peck Lee is a professor at the Department of Software Engineering, University of Malaya. She obtained her PhD degree in computer science from Université Paris 1 Panthéon-Sorbonne. Her current research interests include object-oriented techniques and CASE tools, software reuse, requirements engineering and software quality. She is a member of IEEE and currently in several experts review panels, both locally and internationally.