Scalability in Analysis of Software Architecture

Michael E. Shin
Dept. of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
Michael.Shin@ttu.edu

Bosah Chukwuogo
Dept. of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
bosah.chukwuogo@ttu.edu

Marta E. Calderón
Computer Science Department
University of Costa Rica
San Pedro, Costa Rica
mcalderon@ecci.ucr.cr

Abstract

This paper describes the scalability issue in the analysis of the software architecture with the Colored Petri Net (CPN), which is transformed from the Unified Modeling Language (UML) based software architecture for large-scale application systems. In this paper, the software architecture for an application system is modeled using the UML first, and then it is transformed into a hierarchical CPN model that is executable. Dynamic properties, such as deadlock, of the software architecture for a system are analyzed using the executable CPN model at the levels of different abstraction of components so as to reduce the complexity of the analysis. The ATM system is used to demonstrate the approach.

1. Introduction

As the Unified Modeling Language (UML) [Booch05, Rumbaugh05] becomes an industry wide standardized notation for object-oriented software development, the software architectures for application systems have been specified in the UML notation. The UML-based software architectures for applications need to be analyzed in terms of dynamic behavior of the systems, such as correctness and performance. However, the UML-based software architecture models are not executable so that the dynamic properties can not be analyzed.

The software architectures with the UML for application systems have been transformed into the Colored Petri Net (CPN) [Kristensen98] models in order to evaluate dynamic behavior of the systems. This is because the CPN models can be evaluated by powerful analysis tools, such as the CPNTools [CPNTools07] and Design/CPN [DesignCPN04]. The CPN analysis tools can quickly not only show that a CPN model behaves in the desired manner, but also check several properties, such as boundedness properties, home and liveness properties, and fairness properties. In particular, the concurrency properties of applications, such as deadlock, are evaluated using the occurrence graph and state space report generated by the CPN analysis tools. The occurrence graph traces all possible system states that can be reached by the system so as to check whether the system frees from deadlock. A deadlock-free system ensures that the system is not suspended indefinitely.

However, the CPN analysis tools may not be fit or may fail to analyze large-scale application systems within a reasonable time due to the limited resource. The CPN analysis tools generate all the possible system states for a CPN model, and trace them to check the desired properties. As a system becomes larger, the tools need to generate and trace more system states. This makes it difficult for the tools to finish the analysis within a reasonable time or fall into failure of the analysis. Thus, scalability of the analysis of CPN models needs to be considered in transforming the large-scale software architectural models to the corresponding CPN models.

This paper describes development of the CPN model transformed from the software architecture model with the UML for a large-scale application system by considering the scalability of the analysis of the CPN model. The software architecture for a system, modeled with the UML, is evaluated using the CPN model in terms of the concurrency properties such as deadlock as well as the correctness of message communication among concurrent components constituting the software architecture as well as concurrent objects supporting each component. To do this, the software architecture for a system, modeled with the UML, is transformed into the corresponding software architecture, modeled with the CPN, which has a hierarchical structure in order to reduce the complexity of flat structure. In addition, the software architecture for a system is analyzed using the transformed CPN model at the levels of different abstraction of components.

This paper begins by describing the related work in section 2. Section 3 describes the overview of our approach. Section 4 describes the message communication for software architecture. Section 5 describes the model transformation for software architecture Section 6 describes analysis of software architecture. Section 7 concludes this paper.

2. Related Work

Related work addresses scalability in the analysis of software system models using the CPN model in terms of software requirements and software architecture.

[Baresi97] proposes a technique based on a formalism to define the mapping from front-end informal (specification) notations to formal models. The semantics underlying the mapping is defined by means of sets of rules, and can be tailored to the working environment. The approach proposed in [Baresi97] is concerned with customizability of structured analysis-oriented models, thus allowing semantics to be adapted to the needs of the application systems.

[Elkoutbi98, Elkoutbi00] presents the CPN model transformed from the UML model as two levels of abstraction – the use case level and scenario level. Each use case is mapped to a place at the use case level that has a special place, "Begin," modeling the entry of the system. A use case is decomposed into sub-use cases using "use" relationships among use cases in the use case model. For each decomposed use case, a table is constructed at the scenario level to describe object states associated with scenarios, and then a whole CPN model is produced by merging all scenarios of each use case using an algorithm for scenario integration.

In [Saldhana00, Saldhana01], an Object Petri Nets model of a system is developed from the statechart model and collaboration model of the UML. The system is specified in the center of the statechart model where the statechart model describes states and their transitions in response to events to objects. States in the statechart model are mapped onto Petri Net places and state transitions are mapped onto Petri Net transitions. The collaboration model connects the execution of statecharts of objects. In this approach, modeling systems using the UML is similar to that in the Real-Time Object-Oriented Modeling (ROOM) [Selic94] and Rational Rose Real-Time [Rose05] where each object executes its statechart.

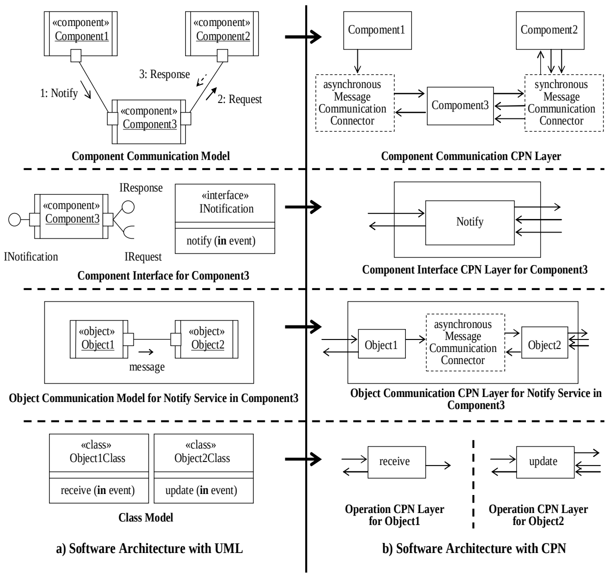
[Pettit00, Pettit04, Pettit06] describes an approach to integrating CPN with software architectural designs created with the COMET method [Gomaa00] and specified in the UML. The behavior of the system is described in the UML collaboration diagrams supported by objects where each object is classified using a stereotype. The behavior of each different type of object in the collaboration diagrams is represented as a behavioral pattern using the CPN.

The approaches above have focused on the transformation of system models to the CPN models so that the system can be analyzed using the CPN models. However, less attention is paid to scalability issues [Shin05] related to model transformation for large-scale systems and the analysis of CPN models.

3. Overview of Approach

Using the UML notation, the software architecture [Buschmann96, Shaw96] for a software system is structured into components and interactions between the components. The UML is an industry-standardized notation for development of object-oriented software systems. A component provides functional services, which are relatively independent of the functionality provided by other components. The interactions between components describe the synchronization in the message communication between components.

The software architecture for a system, modeled in the UML, is transformed into the hierarchical CPN model where a higher level transition (i.e., an activity that transforms data values) is hierarchically decomposed into lower level CPNs. The fundamental notion of the hierarchical structure in CPN models is to reduce complexity of analysis as well as complexity of plain CPN models of large-scale software systems. Fig. 1 depicts the overview of model transformation of the software architecture with the UML ((a) of Fig. 1) to the software architecture with the CPN ((b) of Fig. 1), which is hierarchically transformed into the component communication CPN layer, component interface CPN layer, object communication CPN layer, and operation CPN layer. The CPN model is used to analyze the software architecture for a system. The detailed transformation of the software architecture with the UML to the corresponding CPN model is described in section 5.



«component» Component1
«component» Component2
3: Response
1: Notify
2: Request
«component» Component3
Component Communication Model
Compoment1
Compoment2
asynchronous Message Communication Connector
Compoment3
synchronous Message Communication Connector
Component Communication CPN Layer
IResponse
«component» Component3
INotification
IRequest
«interface» INotification
notify (in event)
Component Interface for Component3
Notify
Component Interface CPN Layer for Component3
«object» Object1
«object» Object2
message
Object Communication Model for Notify Service in Component3
Object1
asynchronous Message Communication Connector
Object2
Object Communication CPN Layer for Notify Service in Component3
«class» Object1Class
receive (in event)
«class» Object2Class
update (in event)
Class Model
receive
update
Operation CPN Layer for Object1
Operation CPN Layer for Object2
a) Software Architecture with UML
b) Software Architecture with CPN

Fig. 1 Overview of Model Transformation

By consideration of the scalability in the analysis of the software architecture for a large-scaled software system, the software architecture is analyzed using the CPN model at the levels of different abstraction of components. The functionality of software architecture is analyzed at the (abstract) component level ((a) in Fig.2) in which the analysis focuses on the interaction between components modeled with less detailed (internal) objects supporting the services provided by the components. Each component is a concurrent component, which communicates with other concurrent components via synchronized message communications. Each component in the software architecture with the UML is mapped to a (component) transition in the CPN model in which the transition needs to be decomposed into a (leaf) sub-net. This is because the CPN model should be executable. For this, the functions provided by each component are modeled with the minimum number of objects supporting the functionality, and then the interaction between the components is analyzed in terms of system properties such as deadlock. The Component1, Component2 and Component3 ((a) in Fig.2) are analyzed at the (abstract) component level.

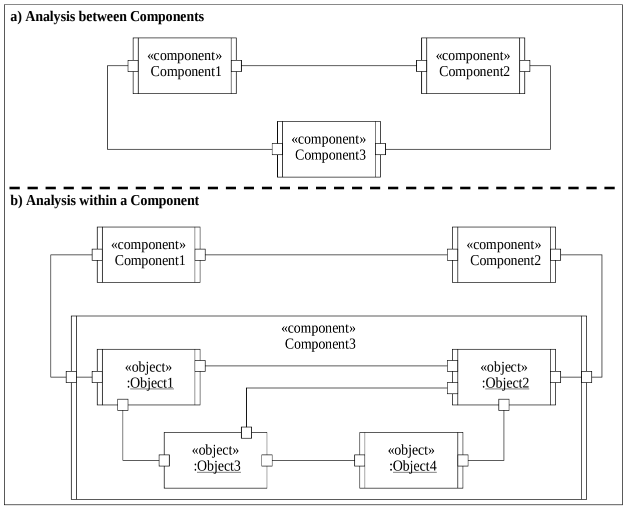


Fig. 2 Analysis of Software Architecture

The detailed interaction between objects within a selected component is analyzed at the (concrete) component level ((b) of Fig. 2). By consideration of scalability of CPN model analysis, a selected component is modeled with detailed objects supporting the component whereas the other (not-selected) components are not changed as modeled at the abstract component level (i.e., the not-selected components were modeled with the minimal objects at the abstract component level). Suppose the Component3 is selected to be analyzed. The services realized by the detailed objects, Object1, Object2, Object3 and Object4, for the selected Component3 ((b) in Fig. 2) are analyzed together with the Component1 and Component2 that are used at the abstract component level. When all the services in the Component3 are analyzed, either Component1 or Component2 is selected to be analyzed. If the Component1 is selected, the Component1 is modeled with the detailed objects, and analyzed with the Component2 and Component3 used at the abstract component level.

4. Message Communication for Software Architecture

The concurrent components or objects in the software architecture communicate with each other in different types of message communication, such as synchronous message communication without reply, synchronous message communication with reply, or asynchronous message communication [Gomaa00]. These types of message communication modeled in the UML notation are transformed to the corresponding message communication connectors in the CPN notation.

With synchronous message communication without reply, a producer sends a message to a consumer and waits for the consumer to receive the message. When the consumer receives the message, the producer can send a new message to the consumer again. Fig. 3 depicts the synchronous message communication without reply using the UML notation ((3a) of Fig. 3), which is transformed into the message buffer connector in the CPN notation ((3b) of Fig. 3) [Pettit06]. A producer sends a message, consisting of both data and control, to a consumer. When the consumer receives the message, it returns the control to the producer. With the control, the producer can send a new message to the consumer.

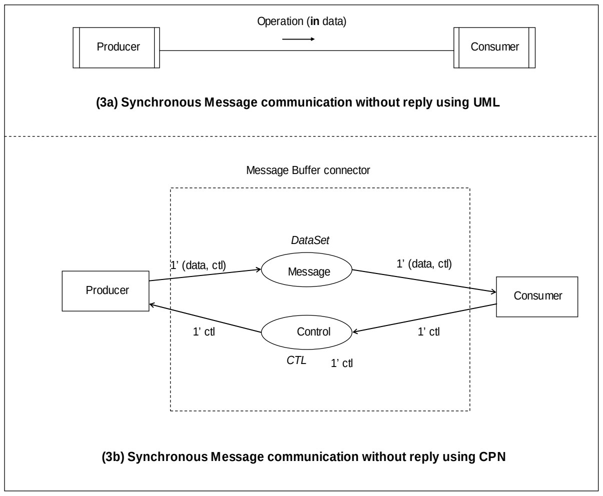
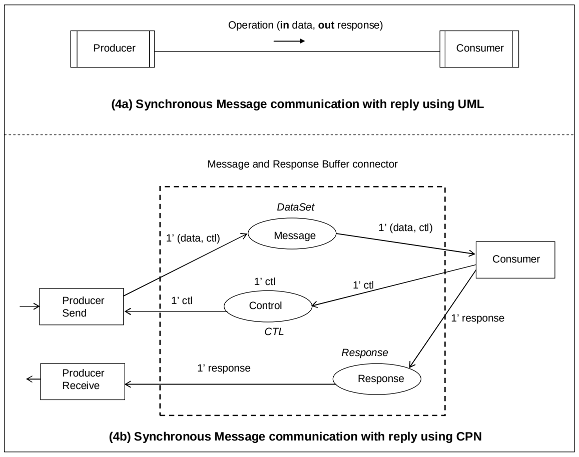


Fig. 3 Synchronous message communication without reply

Synchronous message communication with reply is used to model a client/server message communication. The producer sends a message to the consumer and then waits for a reply from the consumer. Fig. 4 depicts the synchronous message communication with reply using the UML notation ((4a) of Fig. 4), which is transformed into the message and response buffer connector using the CPN notation ((4b) of Fig. 4) [Pettit06]. The producer sends a message with a control to the consumer. The consumer replies a response with a control to the producer. When the response and control arrive at the producer, the producer continues to work.



Operation (in data, out response)
Producer
Consumer
(4a) Synchronous Message communication with reply using UML
Message and Response Buffer connector
DataSet
Message
1' (data, ctl)
1' (data, ctl)
Consumer
1' ctl
1' ctl
1' ctl
Control
CTL
Producer Send
1' response
Response
Response
1' response
Producer Receive
(4b) Synchronous Message communication with reply using CPN

Fig. 4 Synchronous message communication with reply

With asynchronous message communication, the producer sends a message to the consumer and continues to work. Fig. 5 depicts the asynchronous message communication using the UML notation ((5a) of Fig. 5), which is transformed into the message queue connector with size N using the CPN notation ((5b) and (5c) of Fig. 5) [Pettit06]. The producer sends a message, which is stored in a queue (size N) when the consumer does not read a message (i.e., control is empty). The producer can store the next message to the queue until the queue is full. The consumer receives a message from the queue if the queue is not empty.

5. Model Transformation for Software Architecture

The software architecture, modeled using the UML, is transformed into the hierarchical CPN model. The UML communication diagram, referred to as the component communication diagram (top of left-hand side in Fig. 1), is used to model components and their interactions in the software architecture for a software system. This diagram is transformed into the component communication CPN layer (top of right-hand side in Fig. 1) in which a component and an interaction between components in the software architecture model with the UML are mapped to a transition and a message communication connector in the software architecture model with the CPN, respectively. The components, Component1, Component2, and Component3, in the UML model (Fig. 1) are mapped to the transitions, Component1, Component2, and Component3, in the CPN model. Both the interaction between Component1 and Component3, and the interaction between Component2 and Component3 in the UML model are mapped to their corresponding connectors in the CPN model as well.

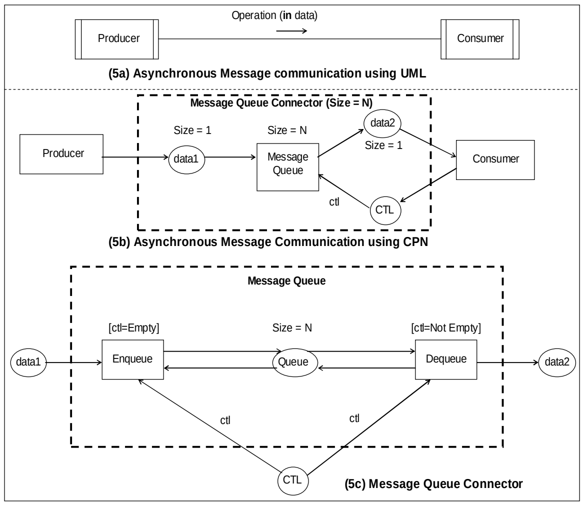


Fig. 5 Asynchronous message communication

Fig. 6 depicts the component communication CPN that is transformed from the UML component communication diagram for the software architecture for the Automated Teller Machine (ATM) system [Gomaa00], which is modified for this paper. The ATM system provides a customer with the three services - withdraw funds, query the balance, and transfer funds - after validating the customer Personal Identification Number (PIN). The software architecture for the ATM system consists of the ATM Client and Server components, having the synchronous message communication between the components.

A component provides other components with functional services and/or requires services from others. The functional services of each component are specified in the interfaces of the component (middle of left-hand side in Fig. 1). The component interface diagram for the software architecture with the UML is transformed into the component interface CPN layer (middle of right-hand side in Fig. 1) where a functional service of a component is mapped to a transition in the sub-CPN of the component transition in the component communication CPN. The Notification is one of Component3 interfaces, defining the “Notify” service, which is mapped to the “Notify” transition in the sub-CPN of the Component3 transition.

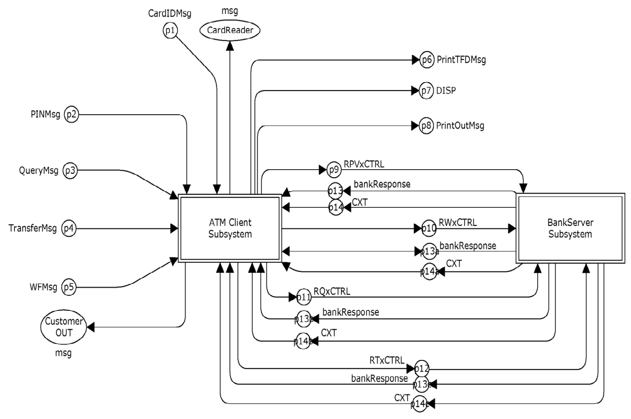


Fig. 6 Component Communication CPN for ATM system

Each service in the interface of a component of the software architecture with the UML is described by means of objects, such as active objects and passive objects accessed by active objects, and the interaction among the objects. An active object has its own thread of control, while a passive object has no thread of control. As with the interaction between components, active objects communicate with each other via the synchronization mechanism. The collaboration among objects for a functional service of an interface of a component is represented using the UML communication diagram (middle of left-hand side in Fig. 1), referred to as the object communication diagram, which is transformed into the object communication CPN layer (middle of right-hand side in Fig. 1). The "Notify" service in the Notification interface is realized by the Object1 and Object2, which are mapped to the Object1 and Object2 transitions in the object communication CPN layer. The interaction between Object1 and Object2 is transformed into the message communication connector in the sub-CPN of the "Notify" transition.

At the abstract component level, the services provided by components are modeled with the minimal number of objects, whereas, at the concrete component level, the services provided by a component selected to be analyzed in detail are modeled with the detailed objects. This aims at reducing the complexity of analysis of the CPN model for the software architecture. Fig. 7 depicts the object communication CPN transformed from detailed objects supporting the Validate PIN Service in the ATM Client component. This service is supported by three active objects such as Card Reader Interface, Customer Interface, and ATM Controller, and two passive (entity) objects such as ATM Card and ATM Transaction entity objects.

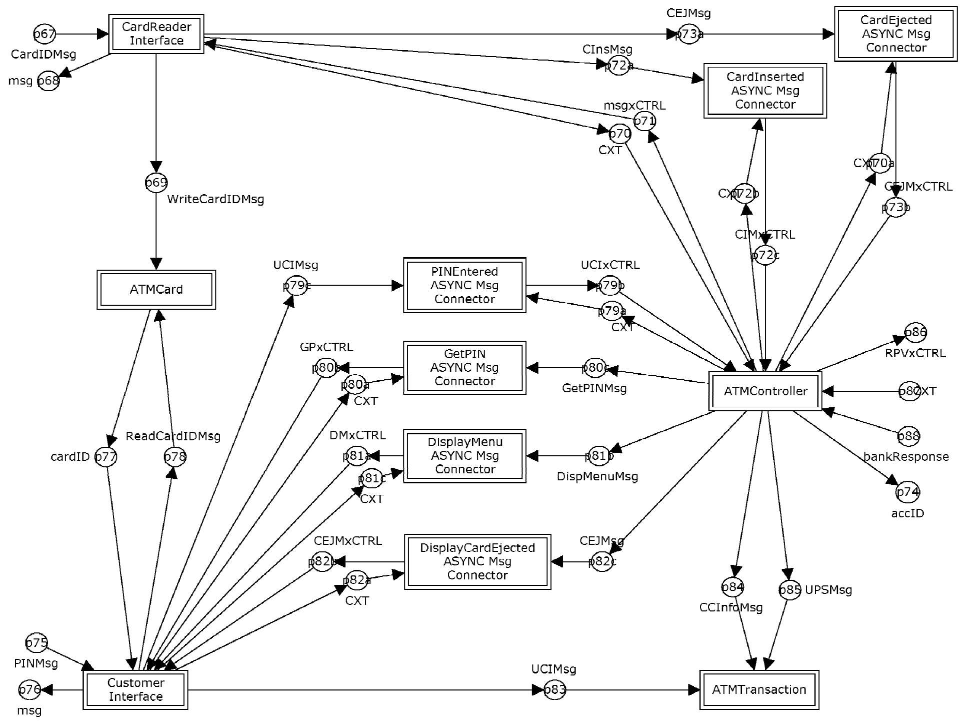


Fig. 7 Detailed Object Communication CPN for Validate PIN Service in ATM Client Component

Fig. 8 depicts the object communication CPN transformed from the minimal number of objects - Card Reader Interface and Customer Interface - for the Validate PIN Service in the ATM client component. The ATM card ID and transaction information are stored directly to the corresponding places in the CPN model (Fig. 8), not through the ATM Card and ATM Transaction passive objects (Fig. 7). The ATM Controller object in Fig. 7 is removed in Fig. 8 by coordinating the execution sequence between the Card Reader Interface and Customer Interface transitions using the data object (token) – Card ID.

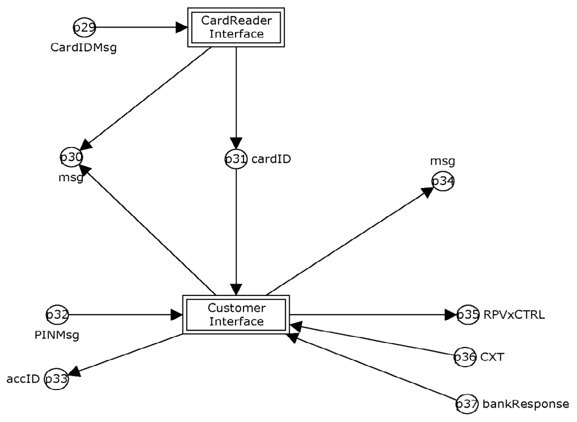
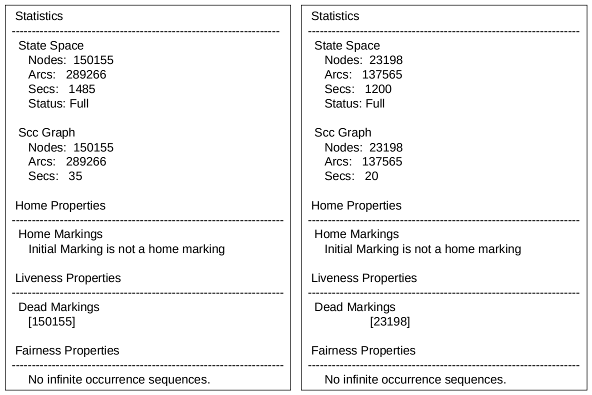


Fig. 8 Abstract Object Communication CPN for Validate PIN Service in ATM Client Components

An object in the object communication model is instantiated from a class in the class model (bottom of left-hand side in Fig. 1), which defines the operations provided by an object (class). When a receiver object receives a message from a sender object, an operation in the receiver object is called so that the object processes the message. The class model for the software architecture with the UML is transformed into the operation CPN layer (bottom of right-hand side in Fig. 1) in which an operation of an object (instantiated from a class) is mapped to a transition in the sub-CPN of a transition in the object communication CPN. When the Object2 receives a message from the Object1 in Fig. 1, the “update” operation of Object2 is called. The “update” operation is mapped to the “update” transition in the sub-CPN of the Object2 transition.

6. Analysis of Software Architecture

The state space reports (Fig. 9) are generated by the CPNTools [CPNTools07] using the occurrence graph of the CPN model for the PIN validation and withdraw funds service provided by the client and server components in the ATM system. The state space report, (a) of Fig. 9, has been generated from the software architecture modeled using the client and server components in which both client and server components are modeled with the detailed objects. On the other hand, in the state space report, (b) of Fig. 9, the client component is modeled with the detailed objects, while the server component is modeled with the minimal objects i.e., abstract object communication diagram.



```
Statistics
------------------------------------------------------------
 State Space
    Nodes:  150155
    Arcs:   289266
    Secs:   1485
    Status: Full

 Scc Graph
    Nodes:  150155
    Arcs:   289266
    Secs:   35

Home Properties
------------------------------------------------------------
 Home Markings
    Initial Marking is not a home marking

Liveness Properties
------------------------------------------------------------
 Dead Markings
    [150155]

Fairness Properties
------------------------------------------------------------
    No infinite occurrence sequences.
```

(a) State space report for concrete ATM client and ATM server components

```
Statistics
------------------------------------------------------------
 State Space
    Nodes:  23198
    Arcs:   137565
    Secs:   1200
    Status: Full

 Scc Graph
    Nodes:  23198
    Arcs:   137565
    Secs:   20

Home Properties
------------------------------------------------------------
 Home Markings
    Initial Marking is not a home marking

Liveness Properties
------------------------------------------------------------
 Dead Markings
             [23198]

Fairness Properties
------------------------------------------------------------
    No infinite occurrence sequences.
```

(b) State space report for concrete ATM client and abstract ATM server components

Fig. 9. State Space Report for ATM System

From the statistics it can be seen that the occurrence graph in (b) of Fig. 9 takes less time in analysis of the model than that in (a) of Fig. 9. The former has 23198 states in the state space report and the latter has 150155 states. More states in a CPN model means that it takes more time in analysis. However, the effects of analysis in both approaches are the same. In (b) of Fig. 9, the software architecture is analyzed using all abstract components, that is, abstract client and server components first, and then each concrete component is analyzed in turn in detail.

The state space reports also show other analysis such as Home properties, Liveness properties, Fairness properties, and even other customized queries that can be defined using ML functions [Jensen02].

7. Conclusions

This paper has described the model transformation of the software architecture with the UML to the corresponding CPN model, which can be used to analyze the dynamic properties of large-scale systems. In order to analyze the CPN models for large-scale application systems within a reasonable time, the software architecture for a system, modeled in the UML, is transformed into the hierarchical CPN model structured into the component communication CPN layer, component interface CPN layer, object communication

CPN layer, and operation CPN layer. The software architecture for a system is analyzed using the CPN model at the levels of different abstraction of components so as to reduce the complexity of the analysis of the CPN model.

This research has the future work. The approach suggested in this paper needs to be validated with more application systems. Currently this approach has been applied to the ATM system. Also this approach can be extended to tool support for the model transformation. The UML-based software architecture model may be captured using a CASE tool such as Rational Rose [Rose05], while the corresponding CPN model is described using the CPNTools [CPNTools07]. In the tool support, the UML model represented using a CASE tool needs to be automatically mapped to the CPN model described using the CPNTools.

References

[Baresi97] L. Baresi, A. Orso, and M. Pezzè, "Introducing Formal Specification Methods in Industrial Practice," In Proceedings of the 1997 International Conference on Software Engineering, pages 56-66, ACM Press, Boston (USA), May 1997.

[Booch05] G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide", Second Edition, Addison Wesley, Reading MA, 2005.

[Buschmann96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, "Pattern Oriented Software Architecture: A System of Patterns," John Wiley & Sons, 1996.

[CPNTools07] http://www.daimi.au.dk/cpntools, "CPNTools on-line," 2007.

[DesignCPN04] http://www.daimi.au.dk/designCPN, "Design/CPN on-line," 2004.

[Elkoutbi00] Mohammed Elkoutbi and Rudolf K. Keller, "User Interface Prototyping based on UML Scenarios and High-level Petri Nets," 21st International Conference on Application and Theory of Petri Nets, Aarhus, Denmark, June 26-30, 2000.

[Elkoutbi98] Mohammed Elkoutbi and Rudolf K. Keller, "Modeling Interactive Systems with Hierarchical Colored Petri Nets," Advanced Simulation Technologies Conference, Boston Park Plaza Hotel, April 5-9, USA, 1998.

[Gomaa00] Hassan Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison-Wesley, 2000.

[Jensen02] Kurt Jensen, Søren Christensen, and Lars M Kristensen, "CPN Tools State Space Manual," University of Aarhus, Aarhus N, Denmark, 2002.

[Kristensen98] Lars M. Kristensen, Soren Christensen, and Kurt Jensen, "The practitioner's guide to colored Petri nets," Internaltion Journal STTT, Vol.2, pages 98-132, 1998.

[Pettit00] R. Pettit and H. Gomaa, "Validation of Dynamic Behavior in UML Using Colored Petri Nets", Proc. Workshop on Dynamic Behaviour in UML Models: Semantic Questions, UML 2000 Conference, York, England, October 2000.

[Pettit04] R. Pettit IV and H. Gomaa, "Modeling Behavioral Patterns of Concurrent Software Architectures Using Petri Nets", *Working IEEE Conference on Software Architectures (WICSA) 2004*, Oslo, Norway, June 2004.

[Pettit06] R. Pettit IV and H. Gomaa, "Modeling Behavioral Design Patterns of Concurrent Objects", *Proc. 28th International Conference on Software Engineering (ICSE)*, Shanghai, China, May 2006.

[Rose05] Rose, http://www-306.ibm.com/software/awdtools/developer/rose/, IBM, 2005.

[Rumbaugh05] J. Rumbaugh, G. Booch, I. Jacobson, "The Unified Modeling Language Reference Manual," Second Edition, Addison Wesley, Reading MA, 2005.

[Saldhana00] John Anil Saldhana and Sol M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," Proceedings of the Int. Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, July 2000.

[Saldhana01] J. Saldhana, S. M. Shatz, and Z. Hu, "Formalization of Object Behavior and Interactions From UML Models," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 11, No. 6, pp. 643-673, Dec. 2001.

[Selic94] B. Selic, G. Gullekson, and P. T. Ward, "Real-Time Object-Oriented Modeling," John Wiley & Sons, 1994.

[Shaw96] M. Shaw and D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline," Prentice Hall, 1996.

[Shin05] M. Shin, A. Levis, L. Wagenhals, and D. Kim, "Analyzing Dynamic Behavior of Large-Scale System through Model Transformation," International Journal of Software Engineering and Knowledge Engineering, Vol. 15, No. 1, 2005, pp. 35-60.