Specifying and Evaluating Software Architectures Based on 4+1 View Model

Kimiyuki FUKUZAWA[1] and Takashi KOBAYASHI[2]
1 Department of Computer Science,
Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
fukuzawa@se.cs.titech.ac.jp
2 Global Scientific Information and Computing Center,
Tokyo Institute of Technology
tkobaya@cs.titech.ac.jp

Abstract: Software Architecture has a great influence on achieving software quality characteristics of a system, so analysis and evaluation techniques are necessary in the early phase of development processes. In this paper, we propose a technique for describing architectures based on 4 + 1 View Model in Rational Unified Process, and a technique for quantifying quality characteristics by transforming an architecture description to Coloured Petri Nets (CPNs) and executing it. To show the effectiveness of our techniques, we describe a RMI architecture based on 4+1 view model by using our description technique and transformed it into a set of CPN. We attach the attribute values of the quality characteristics relevant to security, reliability and efficiency (performance and resource efficiency) to the CPNs and simulated them to validate these quality attributes.

Keywords: Software Architecture, Coloured Petri Net, UML, Quality Charcteristics, Evaluation and Simulation

1. INTRODUCTION

In a software development process, after completing requirements analysis and getting a requirements specification document, a software architecture of the system to be developed is designed. In this sense, architectural design activities play a significant role on bridging a gap between a requirements specification and an implementation of the system. Especially the quality of an

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35614-3_21
C. Rolland et al. (eds.), *Engineering Information Systems in the Internet Context*

architectural design has a great influence on achieving non-functional requirements of the system, such as performance, fault-tolerance, security and so on. In information systems connected over Internet, the achievements of these non-functional factors are one of the most significant issues. If a designer adopted the architecture that was not appropriate for the requirements, the system of low quality would be produced and it would lead to wasting huge amount of cost and development time because he or she has to re-develop the system of a certain quality. To avoid wasting cost and time resulting from the choice of inappropriate architectures, describing an architectural design formally and validating its appropriateness at the architectural design step, before starting the implementation of the system, is very helpful [Medvidovic and Taylor, 1997]. Formal description techniques allow us to evaluate and to validate whether the architecture that was adopted is appropriate for achieving the non-functional requirements or not.

The research community of *software architecture* has paid much attention to designing formal Architecture-Description Languages(ADLs), and some of them are semantically based on Z (Set theory + Predicate Logic) [Abowd et al., 1993], CCS, CSP [Allen and Garlan, 1997],π-calculus [Magee et al., 1998], Chemical Abstract Machine(CHAM) [Inverardi and Wolf, 1997], partially ordered event sets [Luckham and Vera, 1995] and so on.Some of them are combined with queuing models and probability model such as Markov Chain Model so that we could analyze the performance of communication among architectural components by using a simulation technique [Gomaa and Menasce, 2000; Inverardi et al., 1996; Bricconi et al., 2000]. In particular, Architecture Styles (ABASs)[Klein et al., 1998] embeds quality characteristics to a specific architecture style and provides a qualitative reasoning method about behavior of the architecture style. It illustrates the reasoning method based on Markov Chain Model, in order to measure reliability/availability by means of MTTF (mean time to failure). However, the methods to specify software architectures by using these description techniques are not established yet, and it is difficult for practitioners and untrained persons to describe software architectures of practical level formally with these techniques. In addition, we should newly construct a Markov model whenever we try to analyze another architecture or the other quality characteristics other than reliability and performance. On the other hand, in Rational Unified Process (RUP)[Jacobson et al., 1999], 4+1 view model has been developed in order to specify software architectures from multiple viewpoints[Kruchten, 1995], and it was proposed as a method or as a guide on what we should describe as an architectural specification document. However, it has neither specified concrete description technique nor had supporting tools to help architects in describing architectural specifications. The tools for analyzing and for validating architectural descriptions to non-functional requirements have not been developed yet.

The aim of this paper is to propose a method for concretely specifying software architectures of practical level based on 4+1 view model, and a unified technique for analyzing their quality characteristics related to non-functional requirements, differently from the technique where a different Markov model should be constructed for each quality characteristic. To describe software architectures from 4+1 views, we adopt a sub set of UML diagrams and tables. Adopting UML diagrams allows architects to follow a RUP method and to use UML modeling tools like as Rational Rose to describe software architectures.

To analyze an architectural description, we transform it into Coloured Petri Nets (CPNs)[Jensen, 1992] and simulate the CPNs to validate quality attributes of the described architecture. CPN has powerful analysis and simulation techniques and tools such as Design/CPN [Corporation, 1993]. CPN is an extended version of Petri Net, so that computed data values can be carried by the tokens, and we can attach *expressions* to arcs of a net in order to calculate the values on the tokens. The analysis and simulation tools can detect which parts of the architectural design would be the inappropriate ones to the non-functional requirements. In CPN, we can attach the values to tokens, and it allows us to specify behavior of architectural constituents correctly. The quality attributes are presented with numerical values that are on the tokens. Whenever a transition fires and tokens move, the values are re-calculated and updated by evaluating the expressions attached to the arc where the tokens pass. When the execution of the CPN comes to the final state, the resulting quality values are on the tokens. That is to say, executing and simulating the CPNs allows us to calculate the resulting values of the quality attributes. It means that we handle with the only quality characteristics that are closely related behavioral aspects of architectures such as performance, reliability and so on.

The rest of the paper is organized as follows. We start with introducing the overview of CPNs in the next section. In section 3, we summarize the quality characteristics relevant to software architectures and discuss how to express and calculate their values on a CPN. Section 4 presents how to describe a software architecture based on 4+1 view model by using UML diagrams. In section 5, we have a brief sketch of the technique to transform an architectural description into a set of CPNs. Section 6 illustrates a simple case study of RMI (Remote Method Invocation) and discusses the assessment of our technique. We described a RMI architecture based on 4+1 view model by using our description technique and transformed it into a set of CPN. We attached the attribute values of quality characteristics relevant to security, reliability and efficiency (performance and resource efficiency) to the CPNs and simulated them to validate these quality characteristics. This case study shows the effectiveness of our approach.

2. COLOURED PETRI NET (CPN)

Coloured Petri Nets [Jensen, 1992], proposed by K. Jensen, is an extended version of Petri Net. In addition to *places*, *transitions* and *tokens*, the concepts of *colors*, *guards* and *expressions* are introduced so that computed data values can be carried by the tokens. A transition in a CPN is able to fire (called *enabled*) if the following conditions hold:

1 Each of the places input to the transition has at least one token.

2 For the tokens in the input places, the expressions attached to the input arcs to the transition hold.

3 The guard (Boolean expression) attached to the transition holds.

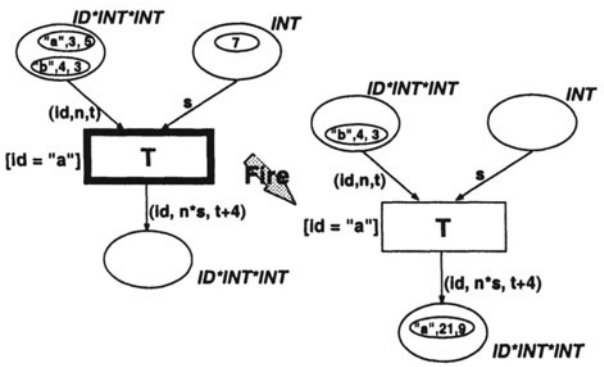

Figure 1. An Example of CPN

Figure 1 illustrates a simple CPN, and ovals and rectangles stand for places and transitions respectively. Tokens put in a place are represented with smaller ovals in the places. The new concept *color* is similar to data type and specifies a set of acceptable data values. Places in a net can be typed, i.e. colors can be attached to the places. For example, suppose that the color "INT" is attached to a place. The tokens in the place cannot have data values except that their type is INT (integer). In the figure, the expressions "(id, n, t)" and "s" are attached to the arc that flows to a transition. The former one stands for a triple of values where one is of ID type and the other two are of INT, and these values are bounded to the variables "id", "n" and "t" respectively. The expression [id = "a"] is a guard of the transition, and since it holds for the token ("a",3), the transition is able to fire. After it fires, the token on the output arc has the value ("a",21,9), which is the result of evaluating the expression (id, n*s, t+4). Suppose that the third element "t"of the value represents time expiration of the

execution. The value "a", 3, 5 on the token at the place expresses that 5 time units are passed until this execution point. The expression "t+4" means that the execution at the transition t consumes 4 time units and we can represent and calculate the quality factor related to performance or time efficiency of the system.

3. QUALITY CHARACTERISTICS AND QUANTIFICATION

3.1. Software Quality Characteristics

In this paper, we adopt quality items as the quality characteristics that have been proposed in ISO/IEC9126[ISO, 1991] to measure the quality of software architectures. This standard classifies software quality into six quality characteristics and 21 quality sub-characteristics as shown in Table 1.

Quality Characteristics	Contents
Functionality	Having functions that users can be satisfied with
Reliability	Maintaining functions under a condition during a period
Usability	Easiness to be used, less efforts to be used
Efficiency	Performance and resource efficiency under a condition
Maintainability	Easiness to be maintained
Portability	Less efforts required to be transferred to another environment

Table 1. Quality Characteristics

As mentioned before, we focus on the behavioral aspects of software architectures and use a CPN technique. Thus we select out the four sub-characteristics, as shown in Table 2 that can be modeled and quantified with CPNs.

Functionality	
Security	Protecting from unauthorized accesses

Reliability	
Maturity	Less potential faults

Efficiency	
Time efficiency	Response time, processing time, throughput
Resource efficiency	The amount of computing resources, available time

Table 2. Adopted Quality Sub-Characteristics

3.2. Security·Maturity·Time Efficiency

Since tokens in a CPN denotes an execution point of an architectural description, the progression of the execution is represented with the movement of the tokens, as shown in Figure 2. To model the quantification of security, maturity and time efficiency, we associate attribute values relevant to them with a token. The value are accumulated and re-calculated as the execution goes by, i.e. the token moves. The expressions for these calculations are attached to arcs in the CPN. We focus on the changes of the values between incoming to and outgoing from each component, because the value change on the component expresses one of its quality attribute values that are made clear by means of its execution. this technique is for calculating quality attribute values for each component. If we want the quality characteristics on the whole of the architecture, we calculate the differences between an initial state and a final state of the execution. We can calculate the average of the quality characteristics. After the execution comes to a final state, we calculate the average of the changes of the values over the components.

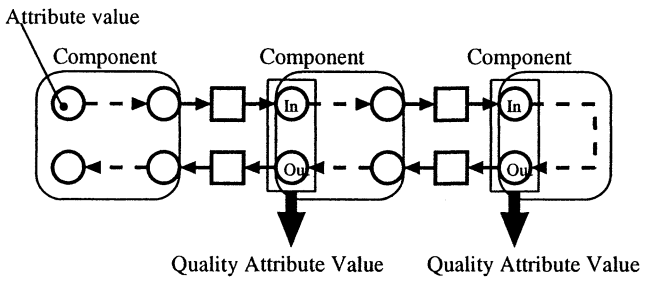

Figure 2. Quantifing Security, Maturity and Time Efficiency

The attribute values relevant to security expresses the degree of risks of information leaks during communications and they depend on 1) the strength of cryptography if we use its technique on the communications and 2) the easiness of wiretapping on the communication lines. In addition, we consider the third factor; the importance of information conveyed on the communication lines. For example, suppose that we send our credit card numbers on a communication line. This information is more important than the name of a card holder. We provide a weighted factor according to the importance of information to calculate the security on the communications.

These three parameters Strength of Cryptography, Easiness of Wiretapping and Importance of Information are provided when we calculate the value of security on a CPN, as a description of implementation view mentioned in section

4. For each transition of a CPN denoting a communication, we incrementally add the following;

$$SV = SV + EW \times \sum_{Transferred\ Data} (II \times (1 - SC))$$

where SV : Security Value, EW : Easiness of Wiretapping, II : Importance of Information and SC : Strength of Cryptography. Note that the two occurrences of SV. The left-hand side stands for a new attribute value of security after a token passes through the transition on a CPN, while the right-hand side is the value before the firing at the transition occurs. That is to say, these represent the value after the communication finishes and the value before the communication respectively. In the case that several types of information are transferred on a communication line, we calculate the summation of $II \times (1 - SC)$ for each data type.

The attribute value related to maturity represents the ratio of successful processing and can be calculated as follows;

$$MV = MV \times SR$$

where MV : Maturity Value and SR : Successful ration on the processing denoted by a transition of a CPN. After finishing the execution of the CPN, we divide the final result of MV by initial MV, i.e. the change of MV, and can get the maturity of the whole of the architecture.

Time efficiency can be defined as a consumption time for the processing and/or the communication. We attach a parameter PT expressing the processing to the corresponding transition of the CPN, and the time consumption is accumulated as the execution goes by.

$$TE = TE + PT$$

where TE : Time Efficiency Value and PT : Consumption Time for processing.

As for time efficiency on a communication, we should consider additional parameters such as the size of transferred data, the throughput on the communication line, the overhead of the used protocol and initialization of the connection for the communication (called set-up time).

$$TE = TE + ST + \left(\sum_{Transferred\ Data} SZ \right) \div TP \times OV$$

where TE : Time Efficiency Value, ST : Set-up time (Time for initializing the connection), SZ : Size of Transferred Data, TP : Throughput and OV : Overhead of the used communication protocol. We calculate the difference

of time efficiency values TE between an initial state and a final state of the execution of the CPN.

As will be mentioned in section 4, these parameters for calculating quality attributes on a CPN are specified as implementation view of an architecture by an architect based on his experiences.

3.3. Resource Efficiency

One of the typical measures expressing resource efficiency is the ratio of the usage of CPUs at a run-time.

In a CPN, a CPU is defined as a token in a place as shown in Figure 3. In the figure, a CPU is allocated at the transition T1 and is released at the transition T2. That is to say, the tokens in the place denote the CPUs that are free. The time during the CPU is allocated is calculated as a difference between the firing time at T1 and at T2.

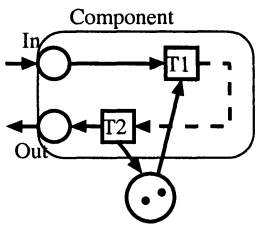

Figure 3. Quantifing Resource Efficiency

$$CT = CT + (T2 - T1)$$

where CT : Time during CPU allocation,
$T2, T1$: Time of CPU allocation and release respectively (firing time at T1 and at T2 in Figure 3).

Resource efficiency value can be calculated as dividing the total time of CT by the number of CPUs.

4. ARCHITECTURAL DESCRIPTION TECHNIQUE

4.1. 4+1 View Model

In 4+1 view model, we model a software architecture from 5 viewpoints; logical view, process view, implementation view, deployment view and use-case view. The benefit of this modeling technique is that we can separate various aspects of software architecture and use the different views depending on the application of the architectural descriptions.

The logical view addresses the structure and the behavior of the system, while the process view describes the concurrency of the architectural constituents in at run-time. The implementation view specifies the modules and the protocols among them that the system uses. We describe how runtime components of the system are allocated to the physical platform or computing nodes in deployment view. Use case view is for helping developers in understanding the descriptions of the other views and describes key scenarios that illustrate typical behavior of the system. In the following sub sections, we illustrate how to describe these 5 views in our approach by using an example of RMI (Remote Method Invocation) architecture.

4.2. Logical View

Since logical view describes both of static structure and behavior of the system, we use a class diagram for the structure and a state diagram for the behavior. Figure 4 is a class diagram for representing the structure of RMI

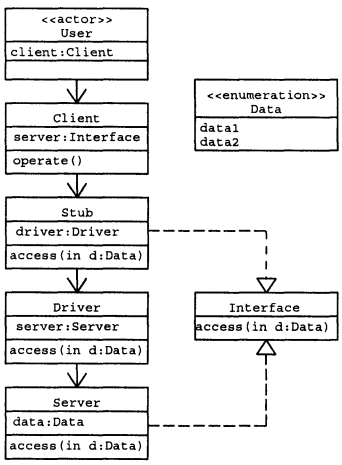

Figure 4. Logical View (Class Diagram)

architecture. An architectural component is defined as a class. In particular, the components that are in the external environment and interact with internal components, e.g. a user, are called actors, and we use the classes having the stereotype <<actor>> to define them. The components can have their attributes and operations. For each operation, we can specify its behavior with a state diagram, and a set of these state diagrams defines the internal behavioral aspect of the system. On the other hand, since an actor has no operations, we define the behavior of an actor with a state diagram. Figure 5 depicts five state

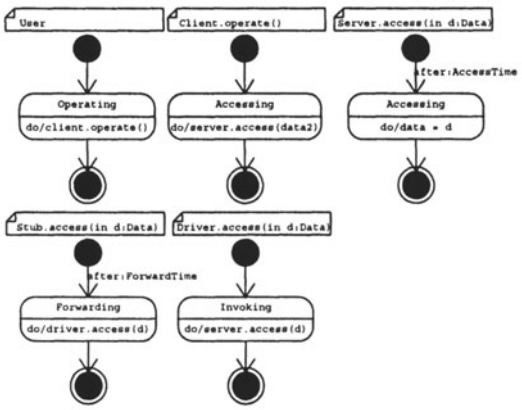

Figure 5. Logical View (State Diagram)

diagrams, one of which specifies the behavior of the actor "User" in figure 4. The other four diagrams define the operations of the architectural components. For example, the leftmost diagram in the top is for the operation "operate" in the class "Client".

In a state diagram, we can describe the various kinds of actions, e.g. updating attribute values, calling an operation in the other component and creating an instance of a component, in a state. Guard conditions that trigger state transitions and/or time expiration expressions can be attached to arcs in a diagram. In the diagram of "access" in "Stub", the time expiration parameter "ForwardTime" and it shows that the transition to state "Forwarding" after ForwardTime passes.

4.3. Process View

A process is a unit of instances of execution in the system for modeling concurrency at run-time and we describe this view in a component diagram. Figure 6 illustrates the process view of RMI architecture.

In the figure, a process is represented with a UML component, which is depicted in a box with two smaller rectangles. A process is also considered as a class whose stereotype is "Process", while an actor is also a class of the stereotype <<Actor>> in the similar way to the logical view.

In the figure, the three processes "ClientProcess", "ServerThread" and "User" are concurrently executed. The UML component denoting a process can contain a set of classes which are executed in the process and these classes should be defined in the class diagram of the logical view description. For example, the two structural constituents of RMI "Client" and "Stub", appearing in Figure 4, are executed in the "ClientProcess".

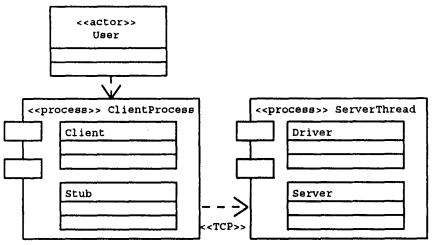

Figure 6. Process View

A relationship between components is depicted with a dashed arrow and we can attach a stereotype to it. The type stands for a communication style such as a communication protocol. In this example, the concurrent process "ClientProcess"communicates to "ServerThread" by using the protocol <<TCP>>.

4.4. Deployment View

In deployment view, we describe a set of hardware components, their communication, the allocation of processes to them and the deployment of the logical components to them, at the starting time of the system operation. As shown in Figure 7, we use UML deployment diagram to define the deployment view of an architecture. In the figure, we have two hardware components; one is "clientHost" and the other is "serverHost", which are connect to each other through <<LAN>>. The clientHost component has one process instance "clientProcess", where the two logical components "c" of Client class and "st" of Stub class are deployed at the starting time of RMI operation.

4.5. Implementation View

We describe the implementation view not with a UML diagram but a table whose entry elements are parameters for calculating quality attribute values.

We have the multiple levels or categories of parameter that can be specified to calculate quality attribute values and list up them in Table 3. This table is used as a template for describing an implementation view. Each category corresponds to the category of description elements from the other views. For example, the category "Data type" is specified in the class diagrams of the logical view, while the elements in "Process communication" are defined as communication protocols in the process view as shown in Figure 6.

The entries in the column "Parameters" are from the parameters for calculating quality attribute values, which were mentioned in sections 3.2 and 3.3.

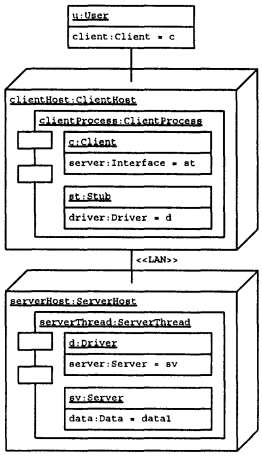

Figure 7. Deployment View

Category	Parameters		
Data Type	Size	Importance of Information	Strength of Cryptography
Operation	Strength of Cryptography	Success ratio	Set-up Time
Process Communication	Overhead		
Hardware Communication	Throughput	Easiness of Wiretapping	Success ratio
Hardware	The number of CPUs		
Time Parameter	Consumption Time		

Table 3. Parameters from Implementation View

In the table, for each data type whose data are passed among the architectural components, we can provide the parameters of the data size, importance of information and strength of cryptography if we use cryptography technique to the data. For the operations which are executed on the components, we can specify as parameters the strength of the cryptography that they use for their input and output data, success ratio of their processing and their set-up time. Since a type of process communication uses some kind of communication protocol such as TCP, the parameter of the overhead of processing the protocol can be attached. The communication among hardware components (hardware communication) includes the quality factors related to the throughput of the communication lines, the easiness of wiretapping of the lines and the success ratio of that transferred

data are not corrupted or lost on the lines. More concretely, we associate these parameters with a stereotype of hardware communication such as <<LAN>> in Figure 7. In hardware category, we specify the number of CPUs and define the consumption time for processing. The consumption time is attached to the transition time in a state diagram specifying the behavior of the logical view.

Table 4 shows an example for the RMI architecture.

Data Type	Size	Importance of Information	Strength of Cryptography
Data	100	1	0

Operations	Strength of Cryptography	Success ratio	Set-up Time
Client.operate	0	1	0
Stub.access	0	1	0
Driver.access	0	1	2
Server.access	0	1	0

Process Communication	Overhead
TCP	1.1

Hardware Communication	Throughput	Easiness of Wiretapping	Success Ratio
LAN	1000	0.9	0.99999

Hardware	The number of CPUs	Time	Consumption Time
clientHost	1	ForwardTime	0.2
serverHost	1	AccessTime	0.1

Table 4. Implementation View

4.6. Use Case View

Although we do not directly use the use case view for quantifying quality characteristics of an architecture, we describe it as a collaboration diagram to improve the understandability of the other view descriptions, as shown in Figure 8.

5. TRANSFORMING DESCRIPTIONS INTO CPNS

This section presents how to integrate the architectural descriptions from different views and how to transform them into CPNs, by using the example of RMI architecture. This architecture is instantiated into the concrete example which has two clients. The transformation is hierarchically done in a top-down approach. As shown in Figure 9, we first transform the deployment description and generate a template whose slots will be filled with CPNs. That is to say, CPNs of the processes are combined into a final CPN. In the figure, the objects are the instances of the classes, e.g. "Client" and "Stub", appearing in a class

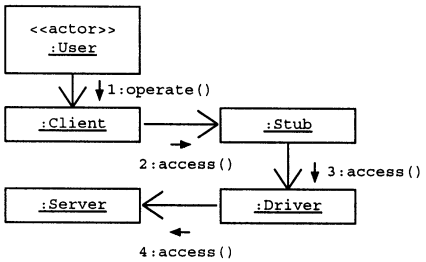

Figure 8. Use Case View

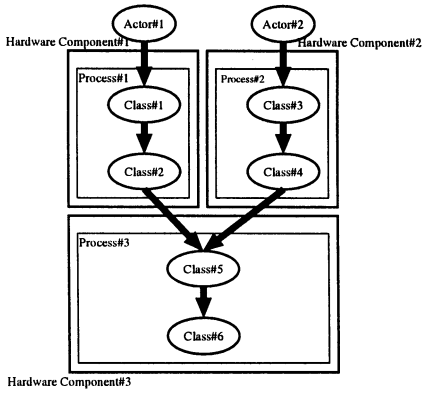

Figure 9. Transformation of Deployment Diagrams

diagram of the logical view, while the processes are specified with a component diagram of the process view, e.g. "ClientProcess" and "ServerThread". Each hardware component such as "ClientHost" and "ServerHost" in Figure 7 is defined in a deployment diagram. We focus on the operation invocation relationships in the logical view to establish the connections of the hardware components that are represented with arrows in Figure 9. For example, due to Figure 5, the operation "Access" in a Stub object invokes the operation "access" of a Driver object, and the CPNs of these operations are connected. The CPN into which a process is transformed consists of the CPNs expressing the classes included in the process. The CPNs for a class contains the CPNs for the operations of the class and a place which holds tokens expressing the instance objects of the class, i.e. logical component instances. If we have two instances of the Client class, we put two tokens on the place connecting the operations in the class, as shown in the right class in Figure 10. These tokens have object

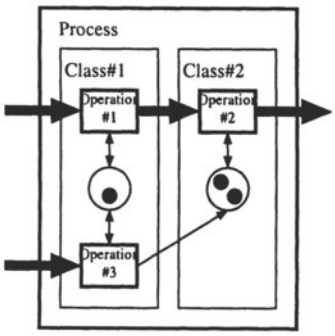

Figure 10. Transformation of Processes

identifiers to discriminate between the instances. When an operation in an logical component instance is being executed, it may access the instance by taking in the corresponding token from the place and it may produce out a new token on the place when generating a new instance. This technique is the same as in [Watanabe et al., 1998] and we can achieve specifying the dynamic generation of architectural components on CPNs.

The CPN expressing an operation is generated from a state diagram in the behavioral description of logical view. Figure 11 sketches how to transform a state transition diagram a CPN. As shown in the figure, each of states and of

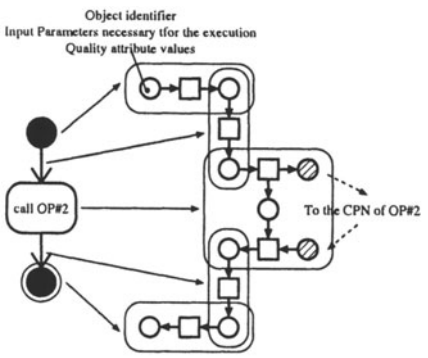

Figure 11. Transformation of State Diagrams

transitions is transformed into a CPNs and they are connected following the connection of states and the transitions. Incoming tokens have the information on the object identifier expressing the executing instance and on the parameters of the operation. In the figure, since another operation is called at the state, the

corresponding CPN has the two places for connecting the CPN of the called operation. We express these places by circles with diagonal lines in the figure. The execution point moves through these places to the CPN corresponding to the called operation.

6. CASE STUDY

To show the effectiveness of our approach, we specify completely the example of RMI architecture based on our approach, transform it into CPNs and simulate them to measure its quality characteristics. Figures 12, 13 and 14 show a page for the whole of the architecture, a page for a process "ClientProcess" and a page for a state diagram of "User" component respectively. We use the deployment view, process view and the implementation view to generate the CPN page for the whole of the architecture, while the CPN pages for the process view is obtained from the process view, the logical view and the implementation view. The descriptions of the logical view and the implementation view are also used to generate the CPN pages for the state diagrams of the logical view.

Table 5 shows the quality attribute values which are calculated by executing the CPNs. This example has just one client process. To compare the quality attribute values, we take another example which contains three clients. In this different example, we change the deployment view description, i.e. we add the additional two process instances of "ClientProcess" in the hardware component "ClientHost" of Figure 7. Its simulation result is shown in Table 6.

	Security	Maturity	Time Efficiency
User	0.9	0.99998	2.41
Client.operate	0.9	0.99998	2.41
Stub.access	0.9	0.99998	2.41
Driver.access	0.0	1.0	0.1
Server.access	0.0	1.0	0.1

	Resource Efficiency
ClientHost	0.083
ServerHost	0.041

Table 5. Quality Attribute Values (Client Process × 1)

This comparative analysis shows that in RMI architecture more clients do not affect on security or maturity, but on time and resource efficiency.

In this section, we had the different configuration of the same architecture type and quantified the quality characteristics. That is to say, we evaluated an architecture with the different configurations. We can take another evaluation style where the different architectures that can achieve the same required functions are evaluated. It allows us to assess their trade-off from the viewpoints of the quality characteristics and to get guidelines to select the most suitable architecture.

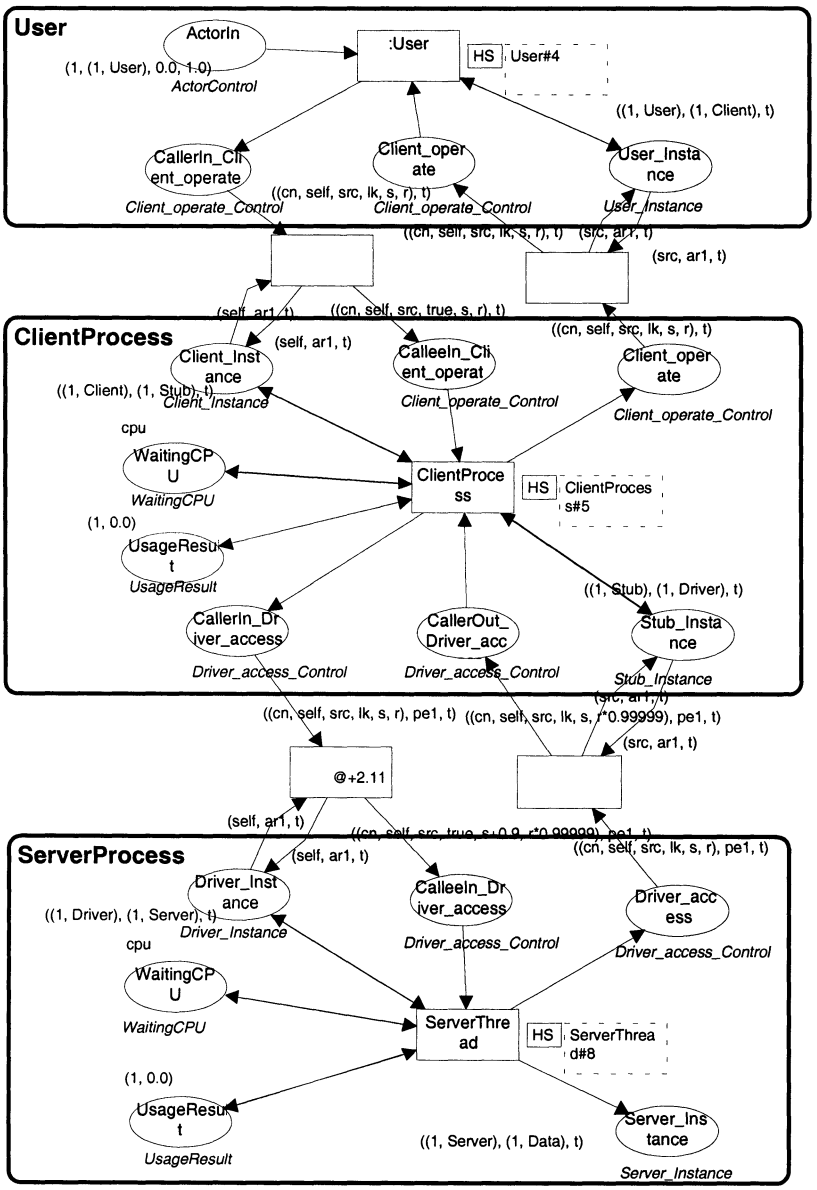

Figure 12. CPN (the whole of the architecture)

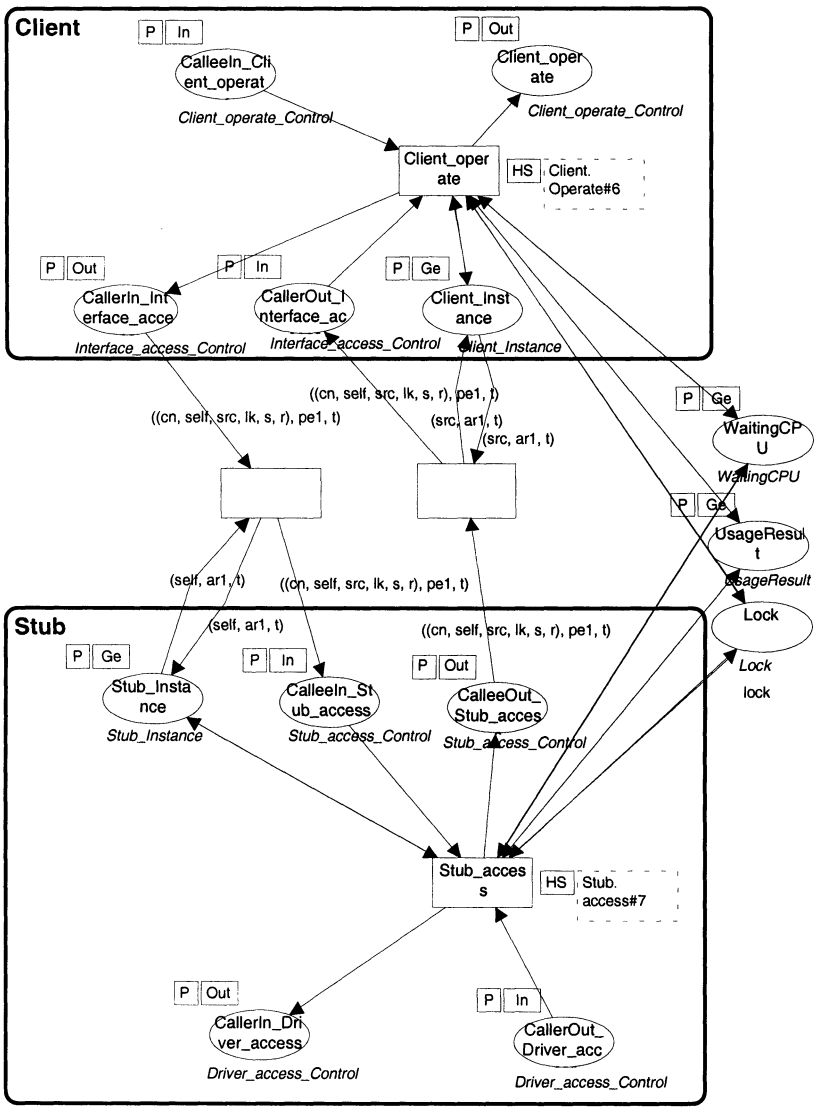

Figure 13. CPN for ClientProcess

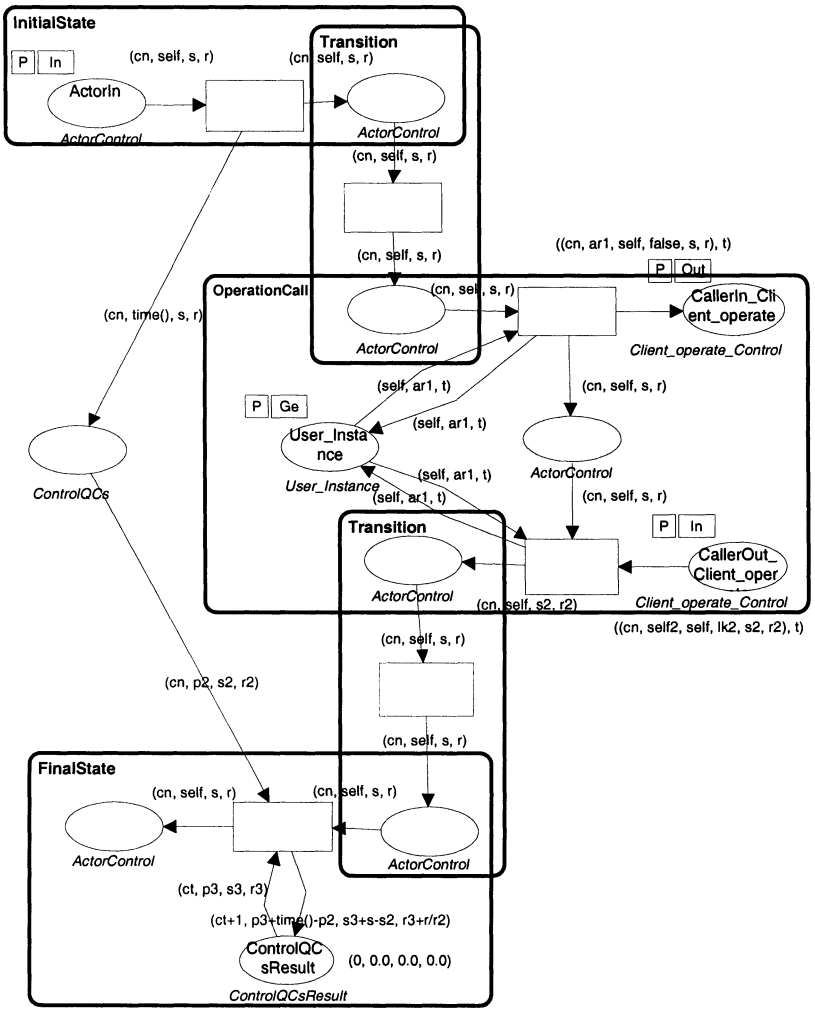

Figure 14. CPN for User Component

	Security	Maturity	Time Efficiency
User	0.9	0.99998	2.51
Client.operate	0.9	0.99998	2.51
Stub.access	0.9	0.99998	2.51
Driver.access	0.0	1.0	0.1
Server.access	0.0	1.0	0.1

	Resource Efficiency
ClientHost	0.077
ServerHost	0.115

Table 6. Quality Attribute Values (Client Process × 3)

7. CONCLUSION

This paper discusses a technique to describe software architectures of practical level based on 4+1view model, and to quantify their quality characteristics by simulating the CPNs into which the descriptions are transformed. And, to assess our technique, we apply it to RMI architecture and analyze its quality characteristics.

We picked up a limited set of quality characteristics, security, efficiency and reliability. However we consider that we can model and evaluate other quality characteristics in our approach, if 1) they can be calculated from some numerical factors, 2) the numerical factors can be assigned to the components and connectors in the architecture, and 3) the calculation can be progressed according the execution of the architecture, i.e. the quality attribute values are decided during executing the architecture. One of the future work is to apply our approach to the other quality characteristics of [ISO, 1991].

One of the future work, and the most important one, is how to formalize dynamism of software architecture like [Egyed and Wile, 2001]. During the execution of the system, some components and/or connectors can be created or deleted. It leads to the change of the structure of its CPN description during the simulation. Additional formalism to CPN is necessary to overcome this issue. The behavioral description based on our ADL is operational in a sense. Embedding declarative descriptions, i.e. describing constraints on behavior is useful to improve the comprehensiveness of the descriptions. We consider the integration of temporal logic formalism with CPNs to achieve the above goal. Exploring the gaps between the architectural description and the implementation, i.e. exploring the techniques how to generate source-code skeletons from the architectural description is also considered to be one of the future work.

ACKNOWLEDGEMENTS

The authors would like to thank anonymous reviewers for their valuable comments to early version of this paper.

REFERENCES

Abowd, G., Allen, R., and Garlan, D. (1993). Using Style to Understand Descriptions of Software Architecture. In *Proceedings of the ACM SIGSOFT '93 Symposium on the Foundations of Software Engineering*, pages 9–20.

Allen, R. and Garlan, D. (1997). A formal basis for architectural connection. In *ACM Transactions on Software Engineering and Methodology*.

Bricconi, G., Di Nitto, E., and Tracanella, E. (2000). Issues in Analyzing the Behavior of Event Dispatching Systems. In *Proc. of 10th International Workshop on Software Specification & Design*, pages 95–103.

Corporation, M. S. (1993). Design/CPN Reference Manual for X-Windows.

Egyed, A. and Wile, D. (2001). Statechart Simulator for Modeling Architectural Dynamics. In *Proc. of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*.

Gomaa, H. and Menasce, D. (2000). Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures. In *Proc. of WOSP 2000*, pages 117–126.

Inverardi, P., Mangano, C., and Balsamo, S. (1996). Performance Evaluation of a Software Architecture : A Case Study. In *Proc. of 8th International Workshop on Software Specification and Design*.

Inverardi, P. and Wolf, A. L. (1997). Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE Trans. Software Eng.*, 21(4).

ISO (1991). Information Technology – Software product evaluation – Quality characteristics and guidelines for their use.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison Wesley.

Jensen, K. (1992). *Coloured Petri Nets*, volume 1-2. Springer-Verlag.

Klein, M., Kazman, R., Bass, L., Carriere, J., Barbacci, M., and Lipson, H. (1998). Attribute-Based Architecture Styles. In *Proc. of WICSA*.

Kruchten, P. B. (1995). The 4 + 1 view model of architecture. *IEEE Software*, 12(6):42–50.

Luckham, D. C. and Vera, J. (1995). An Event-Based Architecture Definition Language. In *IEEE Transactions on Software Engineering*, pages 717–734.

Magee, J., Kramer, J., and Giannakopoulou, D. (1998). Software architecture directed behaviour analysis. In *Proceedings of Ninth International Workshop on Software Specification and Design*.

Medvidovic, N. and Taylor, R. N. (1997). A framework for classifying and comparing architecture description language. *Software Engineering Notes*, 22(6):61–76.

Watanabe, H., Tokuoka, H., Wenxin, W., and Saeki, M. (1998). A technique for analysing and testing object-oriented software using coloured petri nets. In *Proc. of 5th Asia-Pacific Softwrae Engineering Conference (APSEC'98)*, pages 182–190.