

Modeling The Execution Architecture of Mobile Phone Software System by Colored Petri Nets

Jianli Xu

Nokia Research Center

P.O.Box 45

Helsinki, Finland

+358 9 4376 6634

jianli.xu@research.nokia.com

Juha Kuusela

Nokia Research Center

P.O.Box 45

Helsinki, Finland

+358 9 4376 6325

juha.kuusela@research.nokia.com

ABSTRACT

We present an application of Colored Petri Nets for modeling of software architecture. Our work demonstrates how to use architectural modeling to control properties of industrial scale products and product families. The new software architecture of a mobile phone family is modeled by Colored Petri Nets and analyzed using Design/CPN tool. The model describes the execution architecture and allows us to analyze both time and space performance of the software system. Our experience shows that a special purpose model can be constructed also for large systems and this model predicts accurately enough system properties to be useful. The insight this gave us to the internal operation of the software system together with the estimates on its performance guided us through the early phases of architectural design. We continue to maintain the model and expect it to be very valuable tool in configuring different product family members.

Keywords

Software Architecture, Execution Architecture, Modeling, Colored Petri Net, Performance

1 INTRODUCTION

1.1 Architectural Models

Software architecture [1] provides a global view to software system and accordingly software architecture models can be used to predict and control system wide properties. In the ARES¹ project we have built several models of module architecture showing how these models support configuration of variants in a product family [2]. We have also tried several different techniques for modeling execution architectures [3].

Architectural models are used to analyze system wide properties and they have to be complete in respect to that property. Consequently, architectural models often have to be limited to a single viewpoint. The architectural model documented in this paper concentrates on managing dynamic system properties and on predicting system performance.

¹ ARES is supported by the European Commission under ESPRIT framework IV contract #20477 and is pursued by Nokia, ABB, Philips, Imperial College, Technical University of Madrid, and Technical University of Vienna.

From industrial perspective architectural modeling is not yet a mature field. In order to make architectural modeling more useful for software engineers:

- Modeling techniques should be classified according to the reasoning they can support.
- Pre-made models should exist for different architectural constructs allowing analysis of different attributes.
- Models should be composable so that the composite model will correctly predict the properties of the composed system
- Model-checking techniques should be more practical. Models can be partitioned to overcome computational limitations but no guidance exists on how to do it.
- Handbooks on how to model different architectural constructs should exist, currently the results depend entirely on the personal skills of the architect.

Due to various reasons, documented examples of architectural modeling have been small systems. The example used in this paper shows that it is possible to create specific models for analyzing interesting architectural properties of large systems. Several architecture description languages (ADLs) have been developed to model architectures and their properties (see [4]). Here we demonstrate that general purpose modeling techniques like Colored Petri Net[5] and Design/CPN tool[6] can also be applied.

Architectural models share the system structure. The decomposition of the model into parts follows the decomposition of the system and mapping from the entities of the model to the entities of the system is clear. This means that architectural models directly assist in designing and configuring different system variants. Since architectural models share the system structure, it is easy to see how well the model conforms to the system. This increases confidence that the model gives correct predictions.

1.2 Why Colored Petri Net is Chosen for Architectural Modeling

Now in software research community, several common ADLs have been developed to support descriptions for components, connectors, and other aspects of software architecture, like styles, constrains, or design rationale [4][9], they emphasize the structural aspects of the system but are rather weak at describing the performance aspect. In [10] the Unified Modeling Language (UML) [11] is adapted as a ADL in order to integrate the power of ADLs with the day-to-day usefulness of UML.

In designing the architecture of mobile phone software system, we use UML as the ADL for architecture description. Because UML is a standard now and most of our designers can understand it, UML descriptions of the architecture not only provide a standard definition of the system structure and system terminology, but also provide a common and easy language for our designers to communicate with each other. In the architecture descriptions we use UML diagrams instead of those box and line diagrams traditionally used in our software development process.

Designing the architecture of a software system include organizing the system as a composition of components; developing global control structures; selecting protocols for communication, synchronization, and data access; assigning functionality to design elements; physically distributing the components; scaling the system and estimation performance; defining the expected evolutionary paths; and selecting among design alternatives. With UML we can analyze the problem domain and create an architecture model of the solution domain addressing the above architectural aspects, but it is very hard or may be impossible to assess the correctness of our architecture design and how well the designed architecture meets the system requirement. A formal model of the architecture design is needed for specifying and proving the system properties, especially the performance and dynamic properties. Our designers dream about an executable formal model of their architecture design, so that they

can use the model to run different use cases, estimate the performance more accurately, try different system configurations, and select the right design alternatives.

Currently there are many formal modeling methods or languages, such as Z, VDM, CCS, CSP, LOTOS, temporal logic and Petri Nets, just to mention a few. Colored Petri Nets (CPN) have the following advantages over other methods in modeling:

- A CPN model is an intuitive description of the modeled system. It can be used as a specification or presentation. CPN diagrams resemble many of the informal drawings, which are made by designers while they construct and analyze a system.
- CPN supports hierarchical descriptions. This means that we can model large and complex systems in a manageable and compositional way.
- CPN can be extended with time. Therefore we can use the same modeling method for specification and validation of both functional (logical) properties and performance properties.
- CPN has computer tools supporting model building, simulation and formal analysis. Design/CPN is one of the most successful tools, and it has already been used in many practical systems of many application areas.

CPN offers two mechanisms that are of special interest for architectural modeling. First is the flexibility of token definition and manipulation. It is possible to use tokens to model various architectural elements. Second is the concept of sub-page. Sub-pages can be used as a module definition mechanism for various purposes.

Token flexibility comes from the inscription language CPN ML. CPN ML is based on the functional language Standard ML (SML) [7]. Each token in CPN has its own value of a pre-defined data type. Different token types can be used to represent different architectural elements. In our case (see the later parts of this paper), components, tasks, messages, events and even use cases are all described by different types of tokens. The value of tokens can be investigated and modified by the transitions corresponding to different system behaviors.

Sub-pages can be used to create hierarchical system model. The model can be developed either top-down or bottom-up. CPN provides well-defined interfaces between sub-models, and sub-models can be reused. This feature of CPN is very useful in reconfiguring the model in order to analyze alternative policies and mechanisms. It can also be used as a mechanism to define the dimensions of variation in the product family.

The paper is organized as follows. Section 2 is a brief introduction to the mobile phone software system and its architecture design. Section 3 describes how the CPN model is created. Section 4 shows the simulation conducted on the CPN model, and section 5 briefly describes the formal analysis of the model. Section 6 summarizes our experience and results.

2 SOFTWARE SYSTEM OF A NEW FAMILY OF MOBILE PHONES

The product family of Nokia mobile phones consists of a large number of products for each mobile communication standard. As the market has evolved, requirements have changed substantially. The phone has become a part of a distributed system with software running concurrently on handset, Data-card, PC or a car control system. Image and live video transmission have made the real time requirements harder. New requirements also increase the size of the product family and the variation inside the product family.

A new component-based software architecture has been designed to manage the complexity of the product domain and guide the development of new generations of Nokia mobile phones. This architecture defines software components, message interfaces between components, essential use cases, component grouping and deployment structure.

We follow the “4+1” views architecture design approach [12][13] and use UML to describe the architecture. Figure 1 shows the simplified software architecture of the mobile phone family as a UML class diagram (from the logical view). There are three major parts in the architecture: System Objects, Utilities and Communication & Control Kernel. System Objects can be Client objects or Server objects. Client objects interact with users to provide them the custom features of the mobile phone, they are mostly user interface components. Server objects manage and control the functional resources inside the system provide basic system services to Client objects and other Server objects. Examples of resources are call, short message, data call, phone number book, window, energy, and so on. During the execution time, System Objects are grouped into concurrently executing threads called tasks of the operating system. System Objects communicate with each other through the Communicator in Communication & Control Kernel. Communicator is responsible for message routing and both local and remote message passing between System Objects, it has different policies for inter-device and intra-device communication, and different mechanisms for inter-task and intra-task communication. Communication & Control Kernel is also responsible for system object management, task scheduling and event control. Communication & Control Kernel is the infrastructure of the system. Utilities package contains common facilities, such as display and memory management libraries, which can be accessed by all other components. Communication & Control Kernel is generic to all products in the family. Different products may have different configuration of system objects due to different product features and hardware resources.

The new architecture supports the need for different product configurations. However to be able to predict the performance of the system in different configurations before it is implemented, we have to develop a model of the execution architecture.

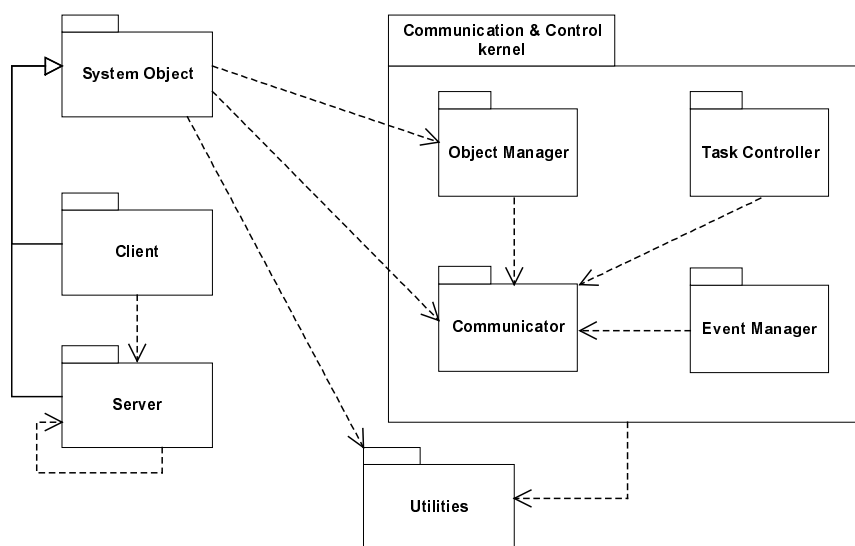


Figure 1. Module architecture of the software system

We used CPN as the modeling language. So far, we have used this model to:

- specify and verify the task control mechanism
- specify and verify the task communication mechanism
- evaluate different task divisions and allocations
- simulate typical use cases
- estimate the message buffer usage and message delays

We started the modeling just after the principal structure of the new software architecture had been outlined and modeling has continued along with the component architecture design and detailed system design. The CPN modeling iterates over three main steps: creating or modifying the model, simulating, and analyzing.

3 CREATING THE MODEL

Before we started to create the CPN model, we had spent about one and a half person months to prepare for the real work, including learning CPN and SML, learning how to use Design/CPN tool, and doing small case studies on Design/CPN tool. In the beginning of the project we used the Design/CPN tool on a Mac-PowerPC with 32Mb RAM, because at the same time we also tried a ADL tool as an backup of UML which only run on Mac-OS. After three months, in order to speed up the simulation and to perform the state space analysis of some real cases, we moved to a SUN workstation (Sun SPARK 2, with 496Mb RAM). Before we moved to the new environment, we had already created the first CPN model. Because Design/CPN tools on these two different platforms do not share any data, we had to spend about two weeks to draw the whole model and test it again on the new platform.

The CPN model was created based on the structure and properties of this architecture. Figure 2 shows the top page of the model. The model contains data objects of devices, functional components (described as client or server), tasks, messages, message queues, and events. Interactions between objects task scheduling and message transmission mechanisms are described using CPN structure and inscriptions. This structure makes it possible to describe task allocation of components, components interactions, space and timing properties of message transmission mechanism, and task control.

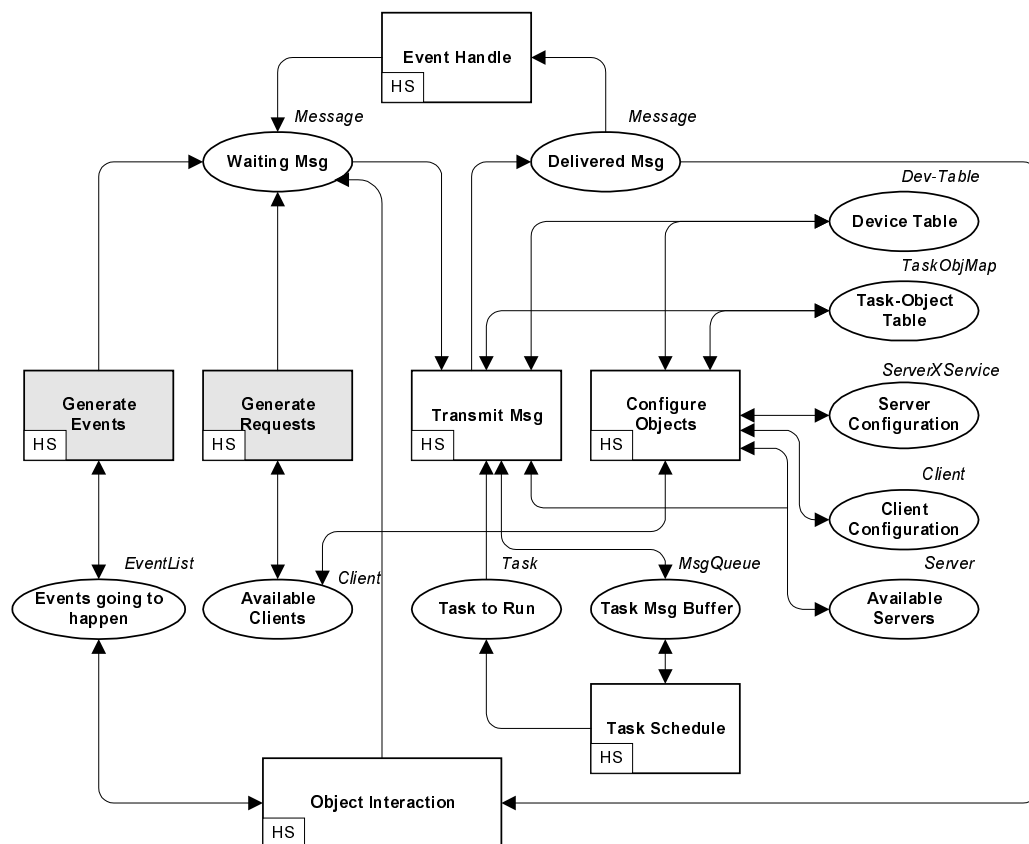


Figure 2. Top page of the CPN model

The mapping between the elements of the module architecture and the elements of the execution architecture is natural. Client and server objects are mapped to Client and Server

type tokens. Object Manager, Task Controller, Event Manager and Communicator modules are mapped to sub-pages Configure Objects, Task Schedule, Event Handle and Transmit Msg. The interactions between system objects are modeled by sub-page Object Interaction.

Since we are not interested in the detailed internal behavior of client or server objects in architectural modeling, client and server objects are abstracted by CPN tokens, which can be reconfigured by giving different initial markings. There are no corresponding components in the module architecture for sub-page Generate Events and Generate Requests. These two sub-pages are added to generate input (use cases) to the CPN model for the simulation purpose.

In order to support the development of an entire product family the model has to be a generic abstraction of the family and it has to be flexible enough to represent different device, component and task configurations. In our model, the part that has to be reconfigured and changed due to differences in different products is described by data variables and initial markings. They can be easily changed without affecting the overall structure of the model. For instance different initial markings in place Device Table represent different hardware configuration, different initial markings in place Task-Object Table give different task divisions, and different initial markings in places Client Configuration and Server Configuration describe different client and server configurations. The generic part of the model is its CPN structure, which represents the properties and mechanisms shared by the products in the family.

The basic system elements (such as client objects, server objects, tasks, devices, service types of each server, and messages between system objects, etc.) are defined by types of tokens. Compositions of the basic system element token types are used to describe higher level tokens, like configuration tables and use cases.

Figure 3 shows the most important token definitions taken from the global definition node of the model. The use cases has been classified into two categories and defined by and respectively. represents the use cases initiated by clients when a mobile phone user starts to use a certain feature, for example, makes a call, answers a call , sends a short message, or check a phone number, etc. represents the use cases started by the system (by server or hardware events), such as indicates an incoming call, indicates a short message received, indicates battery low, etc. Selected use cases of both types are described as initial markings in sub-page Generate Requests and Generate Events before we simulate the execution of them. There UML descriptions of the architecture have defined all the important use cases that fully cover the system functionality, they are collected in the use-case view of the architecture [*]. The use-case view can be used as input for use case definitions in the CPN model.

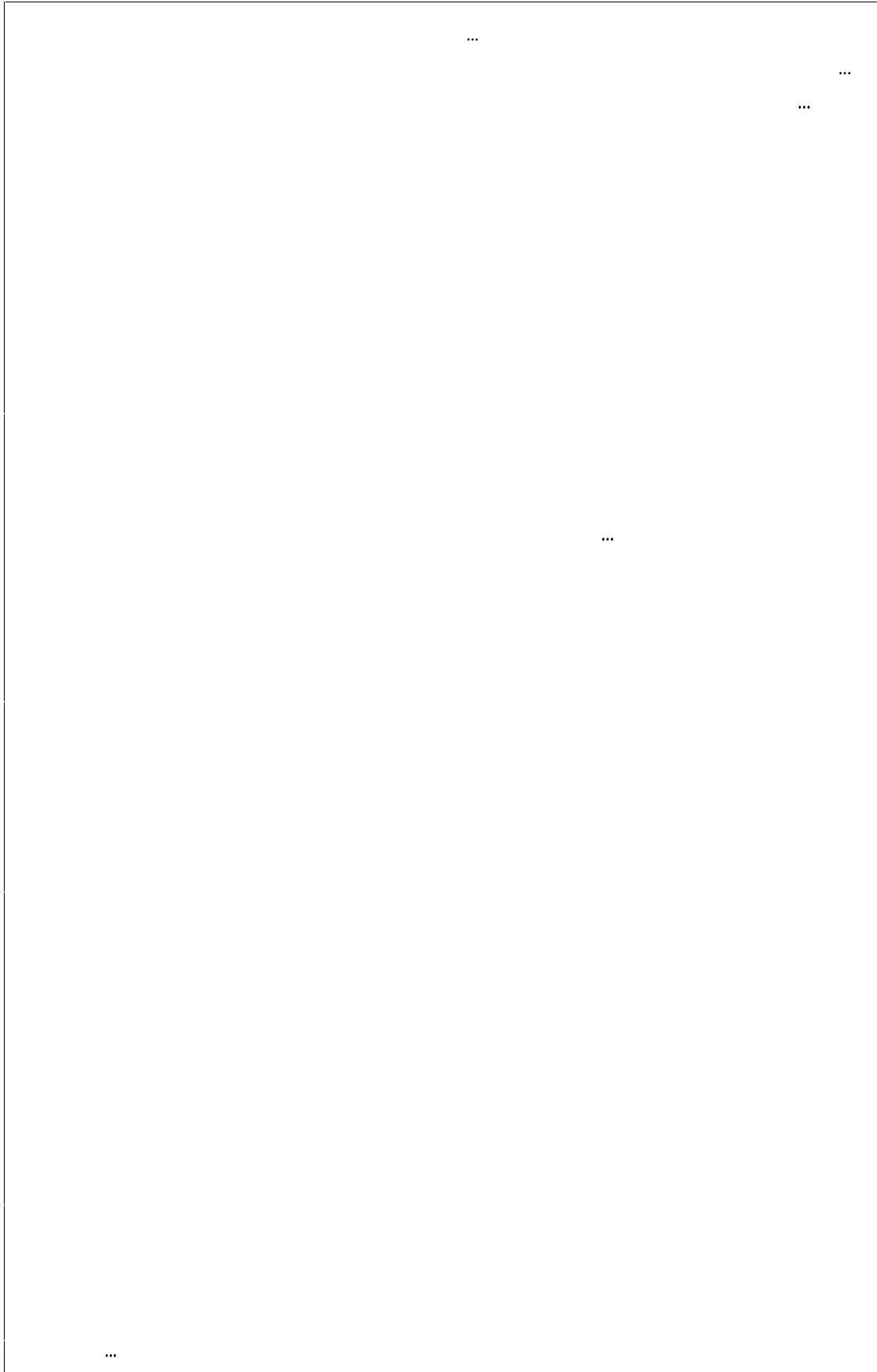


Figure 3 Important token type definitions in the global definition node

Figure 4 shows a simplified example of sub-page Generate Requests. In sub-page Generate Requests, an user may send a request to the system in every based on a probability distribution “ $x_distribution(p)$ ”. The distribution function and the value of “UserReqTime” is decided by the experience and some statistical analysis on our current products. An available client will send a service request to a server when an user request comes. The service request is taken from the use cases predefined in the initial marking of place The initial marking of place ClientUseCase describes the following use cases: a client on a PC (“PcClient”) will send s short message and then add a new item into the phone number book in the handset; a client in the handset (“HandsetClient”) will get a number from the phone number book, make a call by that number, and finally release the call. The service request is converted into a message and is send to the corresponding server. Message routing and transmission is carried out in sub-page Transmit Msg. After the server receives the request message, it will perform the request and send a response message to the client. While the server processing the service request, it may exchange and a few more messages with the client. These client-server interactions are modeled in sub-page Object Interaction.

Similar methods are used for describing and generating type of use cases (in sub-page Generate Events) and hardware signals (in sub-page Wait Signal).

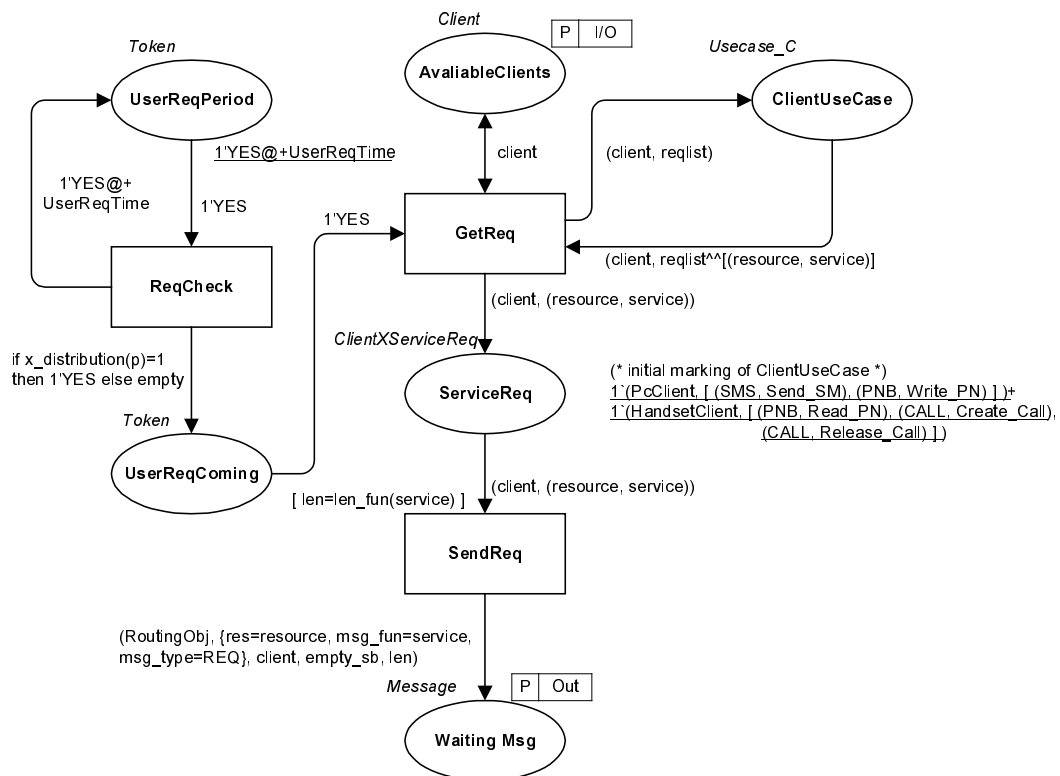


Figure 4 Example of use case description and initiation

In sub-page Object Interaction, in order to focus on architecture properties, we only modeled some most important interactions, such as call handling and short message handling interactions, to a reasonable detail level, and abstracted other interactions just as request-response or indication-confirmation pairs.

The model is hierarchical as shown in Figure 5. Sub-pages (sub-models) in the hierarchy can be replaced or reconfigured. The model building on Mac platform took about one person month. The first model has been changed greatly through simulation to the current model.

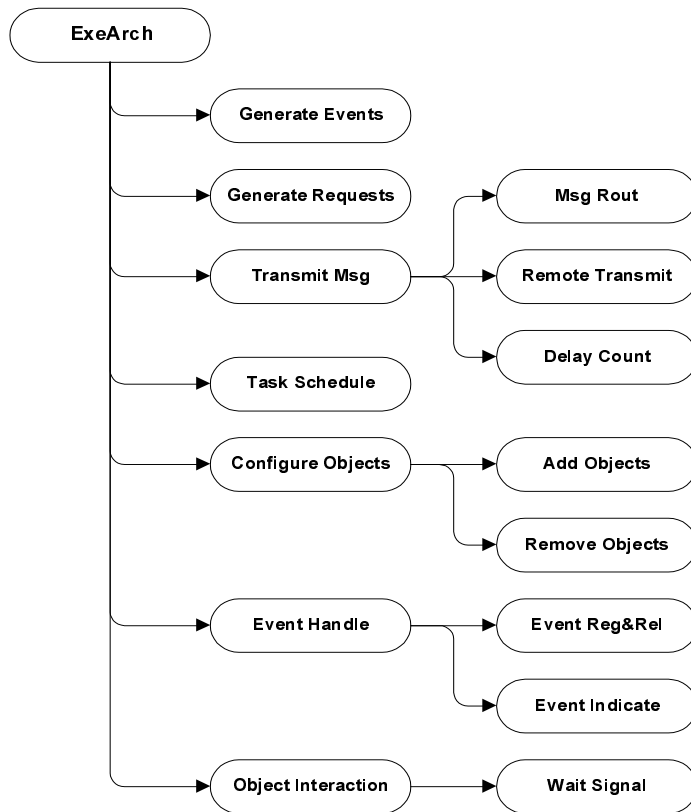


Figure 5. Hierarchy of the CPN model

3.1 Simulation

Deep understanding of the system structure and experience in CPN is required to develop a correct model. As a starting point, we used the component architecture model that shows all components and their interfaces. We structured the execution architecture model by grouping these components into sub-pages. Each sub-page corresponds to a certain class of actions (interactions between components) and a group of actors (components) defined in the module architecture. This structural correspondence helps in validating the model.

Once the kernel part of the model was ready, we ran several use cases. Based on the system specification we know how the system should behave in each use case and we could debug our model. Design/CPN simulator provides interactive simulation, so we can run the model step by step watching the state change and token flow for every step. During this early stage of simulation, we found many inconsistencies. Because this is our first real application of CPN, most of the inconsistencies were errors in the model, but we also found several problems in the original architecture.

After the model was stable, we added time parameters, including message delays on different message links, task-switching time of the operation system, and event processing time. We also defined necessary statistical variables and computations to update them to get quantitative results. We created the input parts (sub-page Generate Requests and Generate Events) of the model to automatically generate the streams of user requests, events and network signals according to different probability distributions. Using these estimates we could compute

- the message buffer usage in the worst case
- minimum, average and maximum message delays (total values)
- and number of task switches needed for each transaction

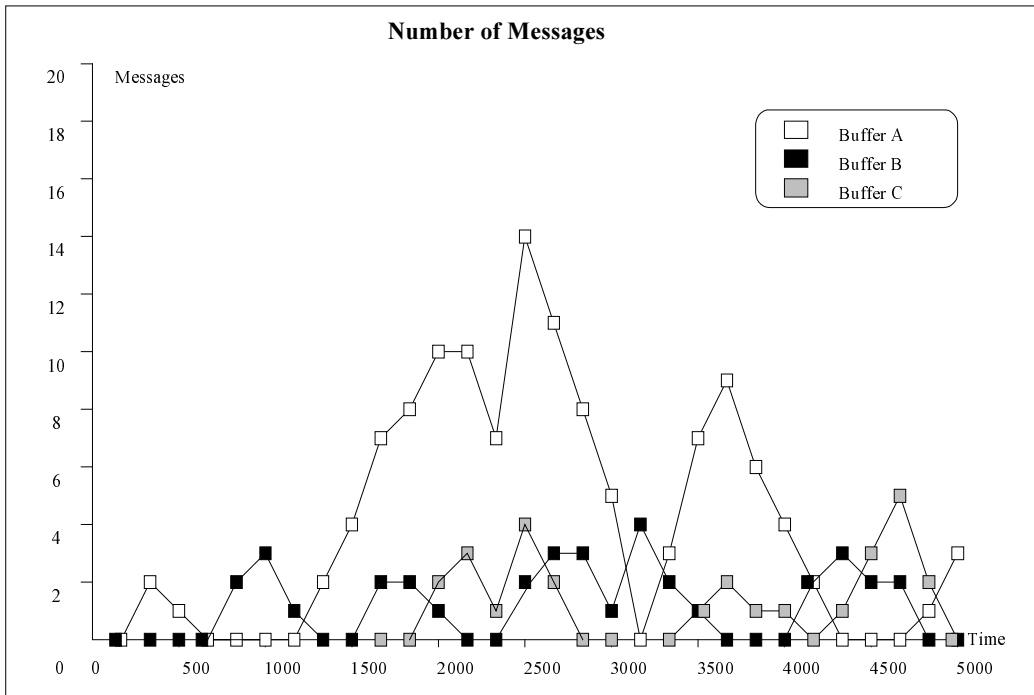


Figure 6. Line chart of message buffer usage

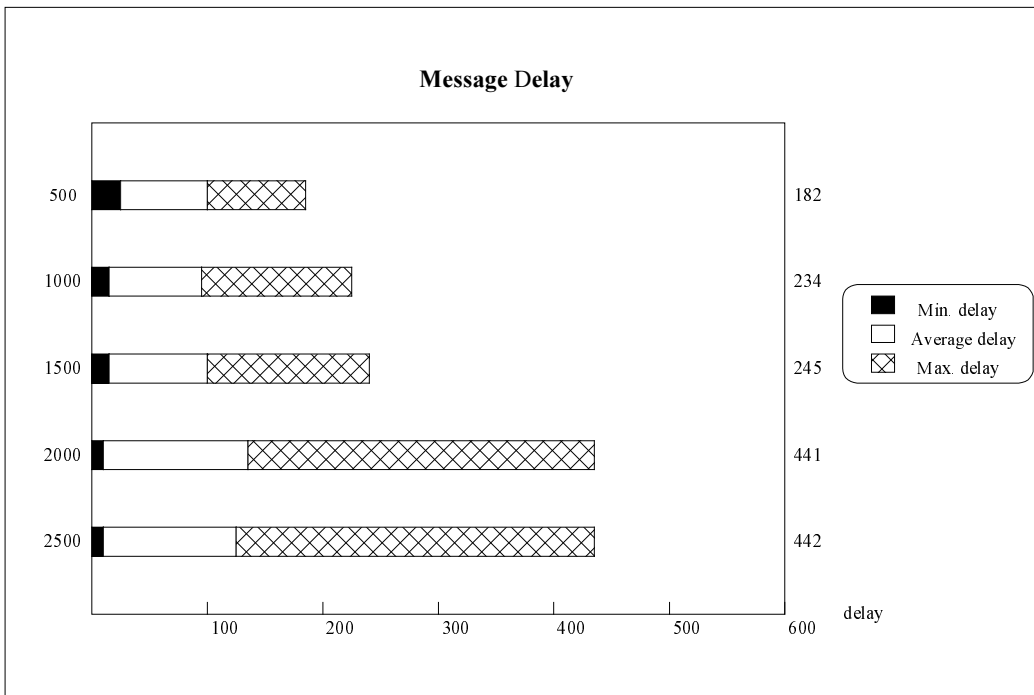


Figure 7. History chart of message delays

Simulation allows us to compare the performance and timing properties of different communication mechanism and control policies. Figure 6 and 7 show the statistic results of a simple use case simulation displayed by chart facilities of the Design/CPN tool. Figure 6 is the line chart of messages in three message buffers of the system at different time, and Figure 7 is a history chart of message delays during the simulation of the same use case, it is updated every 500 time units. From this history chart of message delays we can see the maximum and minimum message delays so far. During or after the simulation, we can accumulate and access statistics about the values of statistic variables generated during the simulation, and show the results by charts or store them in data files.

By simulating a lot of typical use cases and some extreme use cases, we estimated the maximum size of message buffers and the average transaction processing time. Estimated results are based on the actual timing properties of the existing system components and the assumption of the timing properties of new components to be developed.

Sometimes it is necessary to work the other way around. In another project for IP access systems, the total size of packet buffers was determined by hardware configuration. Based on the fixed space requirements, we estimated the timing parameters and used them as timing requirements to the component design and implementation.

We have found some potential problems of the new module architecture through simulation. The communication mechanism may be the bottleneck of the system, overall there are too many task switches, and too many steps in event handling. Description of potential problems together with quantitative analysis of their impact is valuable feedback for the system architect. If these problems emerge in the later phases of implementation, the cost to handle them is not predictable.

3.2 Formal analysis

We did some case studies of formal analysis on the Mac platform. Because of the limitation of processing power and memory restrictions of the platform, we cannot analyze the model as a whole. We have analyzed several important sub-models one by one with the occurrence graph (state space) tool [8]. The tool provides a set of standard queries investigating dynamic properties of the CPN model, such as boundedness, liveness, and home property. Unfortunately, we had to greatly simplify the sub-models in order to get results. We have not done any occurrence graph analysis after we moved to the SUN platform, but we are planning to do it soon.

Now we present an example of occurrence graph analysis. In the initial marking (state) there is one Create_Call request, and one Call_Coming event. That means these two things may happen in any order or even at the same time. One typical scenario of the use case implied by this initial marking is the call collision case, the user is making a call and at the same time, a call comes from the network. The occurrence graph tool successfully generated the full state space after we revised and simplified the model for several times. The standard analysis report of the full state space is given in Appendix, and part of the occurrence graph is shown in figure 8.

In the analysis report, the dead marking is also the home marking. It means that all of the interactions will end at the same state (marking [3659]). We checked this marking in the simulator and found it corresponds to a state where all the service requests and events have been processed and the system is ready to accept new request and event. So we can say that under this initial marking the interaction protocols of call control is correct (no deadlock and safely ended). The analysis report also gives the boundedness properties of every place in the CPN model. For example, under the given initial marking, the maximum number of active tasks is 5 (the upper bound of place ActiveObjects) and the upper bound of the message buffer MsgToBeSent is 5. The cyclic occurrence sequences in those strongly connected components are caused by processing the network signals generated by one transitions which simulates the radio interface to the mobile network.

State space analysis is also an efficient way to debug CPN model. We have found several non-trivial errors in the model during occurrence graph analysis. We found that in most of the time when we could not generate the occurrence graph or SCC, there were some problems in the model, for example unnecessary loops or problematic arc inscriptions. We have not gain many experience of formal analysis so far by just doing small case studies. We are planning to analyze the whole model on the new platform in the near future.

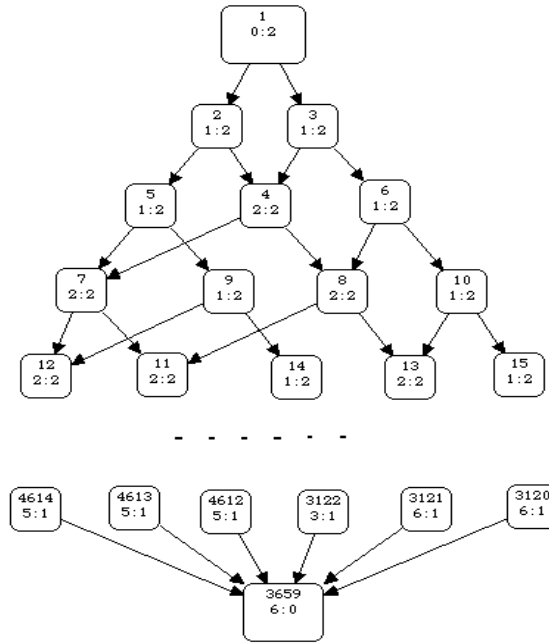


Figure 8. Part of the occurrence graph

4 CONCLUSION

Modeling has been successful. The requirements placed on the system execution architecture have been checked using the CPN model. Alternative mechanisms and policies have been evaluated using the model. We have also found some potential problems. Modeling provides valuable feedback to the development of the architecture. Design/CPN tool has greatly supported our work in designing the new architecture, CPN and Design/CPN tool are fit for this application area quite well if we use them properly.

We have demonstrated that architectural models can be built for industrial systems and general-purpose formal modeling techniques can be applied to architectural modeling. Modeling can provide quantitative results to be used as a guide in developing the system further. Existing tools have limited analytical power but simulation can be used to compensate. The execution architecture model also demonstrates the important benefit of architectural modeling: structural similarity helps in keeping the design and model coherent.

Our main problem was the lack of documentation on how to model software architectures using these techniques. We had to figure out how to model different constructs, where to approximate, how to model the input to the system and how to monitor the behavior. We have now applied the same technique also in modeling another system and it turned out to be a lot easier and much more effective than in the first case. We hope that the documentation of our positive experiences will make architectural modeling a more wide spread activity.

5 ACKNOWLEDGEMENTS

We would like to thank the organizer of this workshop for such a good learning opportunity. We are very grateful to all the reviewers for their constructive comments to our paper.

6 REFERENCES

1. E. Reichtin, M. Maier, "The art of systems architecting", CRC Press, Boca Raton, USA, 1997.
2. A.Ran, J.Kuusela, "Selected Issues in Architecture of Software Intensive Products", Proceedings of ISAW-2, ACM, 1996.
3. Jeff Kramer and Jeff Magee, "Analysing Dynamic Change in Software Architectures: A case study", 4th IEEE International Conference on Configurable Distributed Systems, Annapolis, May 1998
4. Nedad Medvidovic and David S. Rosenblum, "Domains of Concern in Software Architectures and Architecture Description Languages", Proceedings of the USENIX Conference on Domain-Specific Languages, pages 199-212, Santa Barbara, CA, October 15-17, 1997
5. K. Jensen. "Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use", Volume 1. Monogr. in Theor. CS, Springer-Verlag, 1992.
6. Computer Science Department of University of Aarhus and Meta Software Corporation. "Design/CPN Reference Manual", Cambridge, MA, USA. 1993.
7. L. C. Paulson. "ML for the Working Programmer", 2nd Edition. Cambridge University Press, July 1996.
8. Computer Science Department of University of Aarhus, "Design/CPN Occurrence Graph Manual", Version 3.0. Aarhus, Denmark, 1996.
9. M. Shaw, "Truth vs Knowledge: The difference between what a component does and what we know it does", Proceedings of 8th International Workshop on Software Specification and Design, March 1996.
10. J. E. Robbins, and et al, "Integrating Architecture Description Languages with a Standard Design Method", Presented at the Second EDCS Cross Cluster Meeting in Austin, Texas. (can be got from <http://www.ics.uci.edu/pub/arch/sw-and-pubs.shtml>)
11. Rational, "UML Notation Guide", version 1.1, September 1997, <http://www.rational.com/UML>
12. P. Kruchten, "The 4+1 View Model of Software Architecture", IEEE Software, November 1995.
13. H. Eriksson and M. Penker, "UML Toolkit", Wiley Computer Publishing, John Wiley & Sons, Inc., 1998.

7 APPENDIX

Example of state space analysis report

Occurrence Graph

Nodes: 4614
Arcs: 16732
Secs: 1795
Status: Full

Scc Graph

Nodes: 3912
Arcs: 11202
Secs: 35

Boundedness Properties

Best Integers Bounds

	Upper	Lower
CallTrans'ActiveObjects 1	5	0
CallTrans'CallComing 1	1	1
CallTrans'CallCreated 1	2	0
CallTrans'CallMsgForApp 1	5	0
CallTrans'CallMsgForServer 1	2	0
CallTrans'MO_Sl_Finished 1	1	0
CallTrans'MsgDelivered 1	3	3
CallTrans'MsgToBeSent 1	5	0
EventIndicate'ActiveObjects 1	5	0
EventIndicate'CallComing 1	1	1
EventIndicate'EventonGoing 1	1	0
EventIndicate'EventsHappend 1	1	0
EventIndicate'EventsSubscribed 1 1	1	0
EventIndicate'MsgDelivered 1	3	3
EventIndicate'MsgToBeSent 1	5	0
Exec'ActiveObjects 1	5	0
Exec'AvailableServers 1	3	3
Exec'EventsHappend 1	1	0
Exec'EventsSubscribed 1	1	0
Exec'MsgDelivered 1	3	3
Exec'MsgToBeSent 1	5	0
Exec'WantedServices 1	1	0
MsgTransmit'ActiveObjects 1	5	0
MsgTransmit'AvailableServers1	3	3
MsgTransmit'MsgDelivered 1	3	3
MsgTransmit'MsgToBeSent 1	5	0

...

Home Properties

Home Markings: [3659]

Liveness Properties

Dead Markings: [3659]
Dead Transitions Instances: All
Live Transitions Instances: None