Modeling Performance as an Aspect: a UML Based Approach

Kendra Cooper
Department of Computer Science
Mail Station 31
University of Texas at Dallas
Richardson, TX, USA P.O. 830688
972 883 4216
kcooper@utdallas.edu

Lirong Dai*
Department of Computer Science
Mail Station 31
University of Texas at Dallas
Richardson, TX, USA P.O. 830688
972 883 4216
lirongd@utdallas.edu

Yi Deng
School of Computer Science
Florida International
University
Miami, FL, USA 33199
305 348 1831
deng@cs.fiu.edu

Jing Dong
Department of Computer Science
Mail Station 31
University of Texas at Dallas
Richardson, TX, USA P.O. 830688
972 883 2187
jdong@utdallas.edu

ABSTRACT
Non-functional properties, key criteria in determining the success of a software system, need to be addressed early in the software development lifecycle. As these properties interact with, or crosscut, many components and connectors in the architecture of a system, an aspect-oriented design approach appears to be a suitable solution for modeling them. This work presents an overview of our aspect-oriented formal design analysis framework (FDAF) and how it can be used to design and analyze performance properties. Our approach uses a new extension to the real-time UML notation that supports modeling response time performance as an aspect. The extended UML design is manually translated using algorithms into the architectural description language Rapide and analyzed using Rapide's tools for timing simulation and violations. The approach is illustrated using a Domain Name Service (DNS) example. The DNS is a real-time, distributed system.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *object-oriented design methods, computer-aided software engineering (CASE) , modules and interfaces.*

General Terms
Performance, Design, Verification.

Keywords
aspect-oriented, formal methods, UML, Rapide

1. INTRODUCTION
Qualities, such as performance, security, modifiability etc., should be "built" into the system in an early phase (i.e., before it is implemented). The significant benefits of performing such design and analysis work are to detect and remove defects earlier, reduce development costs and time, and improve the quality of the product.

These benefits motivate researchers to investigate software engineering methodologies to support the effective development of large-scale systems that meet their non-functional aspects. Aspect-Oriented Software Development (AOSD) [19], [22] has been proposed as a technology to address this issue. AOSD techniques allow one to modularize crosscutting concerns into separate "aspects" of a system and integrate those aspects with other kinds of modules throughout the software development lifecycle. A property of a system needs to be implemented as an aspect when it cannot be clearly encapsulated in a generalized procedure [15]. In our work, aspects include non-functional properties such as performance, security, adaptability.

A *design aspect* [8] provides a design view that contains only information about a concern realization and it may or may not be implemented in the final product. Clements [6] points out that performance is basically a function of the frequency and nature of inter-component communication, in addition to performance characteristics of those component themselves. Hence, performance concern tends to crosscut a number of components in a system and can be viewed as an aspect.

In this work, we define performance as an aspect. More specifically, we define the performance aspect as a collection of subaspects which include response time, rate throughput, resource

*contact author

utilization, probability, time between errors, durations of event, time between events. These elements have been identified in earlier research in performance analysis [10], [25]. The subaspects are available as a re-usable collection of components [7]. We focus on modeling the response time performance aspect and propose an aspect-oriented extension to the Unified Modeling Language (UML) [5] to model it.

We have chosen real-time UML as the base notation in the approach [20] because it is a well-known semi-formal notation that has been used to describe object-oriented models in the software engineering community. An advantage of using UML is that it is considered to be easier to read and understand than many formal methods. In the effort to improve UML's expressive power to model real-time constraints such as schedulability, performance, and time, the real-time extension has been proposed [20]. Concepts to support performance analysis defined in the extension is adopted by our approach.

One of the main motivations of modeling performance is to evaluate and analyze whether the design meets the system's performance requirements. However, the standard UML lacks such formal analysis mechanism as formal notations do. Several approaches [4], [9] and [23] have been proposed to translate certain UML diagrams into different formal notations in order to analyze performance. An obvious advantage of these approaches is by using formal analytical methods, quantitative analysis can be integrated into software development [2]. Based on the same reason, the UML extension with additional performance aspect information in our approach is translated into Architecture Description Language (ADL) Rapide [16]. ADLs have been developed as formal notations to represent and reason about software architectures. The reason we chose Rapide as the notation in the approach is that Rapide is an event-based concurrent object-oriented language designed for simulation and behavioral analysis of architectures of distributed systems, such as our example system, the Domain Name System (DNS) [17]. Rapide also provides timing model to allow designers to describe and analyze time sensitive prototypes. In addition, analysis tools are available in Rapide and can be used to simulate a software architecture's response time.

The DNS example is used in this paper to illustrate our approach. It has been selected because the DNS is a complex system with a rich set of functional and non-functional requirements. It is real-time, distributed, needs to be secure, and (optionally) supports recursive queries. In addition, the DNS is a non-proprietary standard.

At current stage, the translation from the UML extension into Rapide is done manually. The translation algorithms have been defined and documented; they are going to be implemented in the next step of our work to provide automated tool support.

The paper is organized as follows. Following the Introduction, Section 2 presents the related work. Our performance modeling approached is discussed in Section 3 and performance analysis techniques are presented in Section 4. Section 5 illustrates the DNS application of our approach. Conclusions and future work are presented in Section 6.

2. RELATED WORK

Much attention has been focused on the problem of analyzing a systems' performance during the design phase. Such approaches can be found in [11], [13], [27] and [28]. The Software Architecture Analysis Method (SAAM) [11] is an approach that uses scenarios to drives information about an architecture's ability to meet certain quality objectives such as performance, reliability, or modifiability. Later, SAAM has been extended by the Architecture Tradeoff Analysis Method (ATAM) [13] to consider interactions among quality objectives and identify architectural features that are sensitive to more than one quality attributes. Tradeoffs between quality objectives are evaluated once those sensitivities have been identified. PASA [28] is another method for performance assessment of software architectures. This approach uses the principles and techniques of software performance engineering (SPE) [25] to determine whether an architecture is capable of supporting its performance objectives.

Software performance engineering (SPE) [25] is a method for constructing software systems to meet performance objectives. The SPE process begins early in the software lifecycle and uses quantitative methods to identify satisfactory designs and to eliminate those that are likely to have unacceptable performance, before developers invest significant time in implementation. SPE approaches have been tailored to the UML in [26].

As the UML is a widely accepted notation for system specification because of its easy understandable graphical representations, the UML has been selected to derive systems' performance models from software architecture specifications in several approaches, such as [4], [9] and [23]. All these approaches focus on mapping UML specifications into a certain type of performance model. Three main classes of stochastic performance models are: Stochastic Timed Petri nets (STPN), Queuing Network models (QNM) and Stochastic Process Algebras (SPA). Bernardi et al. [4] proposed an approach to validate and evaluate systems' performance by translating UML sequence diagrams and statecharts into Petri nets. While in [9], a UML model is translated to a queuing network in order to calculate systems' response time and its resource utilization. An approach of using both UML and SPA for performance analysis is described in [23]. In this approach, UML designs are transformed systematically into process algebra models and so, by suitable simulation, used to provide performance estimates.

To support Aspect-Oriented Design, a number of UML extensions have been proposed. Examples of such extensions are [1], [3], and [21]. Aldawud et al. [1] initiated a proposal of a UML profile to support aspect-oriented software development as well as requirements for completing such a profile. In [3], an UML extension is introduced to model aspects with new design element, whose graphical representation is a circle with a cross inside, added into the current existing UML. This notation is used to indicate the join points as it assume that the project is developed using aspect-oriented programming language AspectJ [14]. The approach [21] also presents a UML notation for designing aspect-oriented applications, which is extracted from the concepts defined in the AspectJ language. This notation is proposed for a first step towards a high-level designing graphical language that can be used when building aspect-oriented applications. Three main additional concepts defined are: groups (provide classification means for heterogeneous

and distributed entities), pointcut relationships (allow the programmer to define crosscuts within the functional program), and aspect-classes (actually implement the extension of the program on the crosscutting points denoted by the pointcut relations). Two example aspects, authentication aspect and session aspect are modeled in [21]. As this extension is largely based on AspectJ, it may be more suitable to model aspects (such as security) that need to be implemented in the final product. However, for aspects such as a performance aspect whose modeling purpose is to assist designers to evaluate their design and is not implemented as functional capabilities in code this extension is not sufficient.

3. PERFORMNCE ASPECT MODELING

3.1 Performance Aspect Definition

As described in [10], a performance study needs a set of performance criteria or metrics. One approach to prepare this set is to list the services offered by the system and categorize the possible outcomes into three groups. The first group is the system performs the service correctly. Within this category, the time taken to perform the service, the rate at which the service is performed, and the resource utilization may be measured. The second category is the system does not perform the service correctly. Here, the probability of an error occurring and the time between errors can be measured. The third category is the system cannot perform the service (e.g., the system may be down). In this category, the duration of the event and the time between events can be measured. Based on the study, we defined performance aspect as set of subaspects, which can be mathematically expressed as: Performance Aspect = {Response Time, Rate Throughput, Resource Utilization, Probability, Time Between Errors, Durations Of Event, Time Between Events}. As a system could have its own special requirements, performance subaspects are not limited to those listed. According to the need of a particular application, requirement engineers and designers might define their own performance subaspects of interest and add them into this set.

3.2 Performance Aspect Modeling

The base notation to model performance aspect in our approach is the UML. In this section, a description of how to extend UML to capture performance aspect is presented.

In order to express performance information in a UML model, one has to identify the basic abstractions and relationships that are used in performance analysis. For this purpose, OMG [20] describes a profile that is intended to provide general facilities for capturing performance requirements within the design context, and associating performance-related QoS characteristics with selected elements of a UML model. A general performance model is introduced in the document and its concepts have been adopted in our approach. In [20], a set of performance related stereotypes and their association constraints are defined. Each stereotype may have one or more tags to denote its corresponding performance values. In addition, [20] argues that simple numeric values for performance-related QoS characteristics are not sufficient. Therefore, the concept of performance values is introduced, which gives more information to a value, e.g., a given value may represent an average or maximum, or, it may be a prediction or a measurement.

One of the initial steps of performance analysis of software systems is identifying key performance scenarios [12], [20], [28]. A scenario is a sequence of actions performed by a group of different objects and represents responses with response times and throughputs. QoS requirements are placed on scenarios.

In our UML extension, to make those identified performance related elements in the design diagram obvious to designers, a wedge-like, triangular symbol instead of the UML regular note notation is used. As a UML regular note notation could be used anywhere to record information related to the design and it is not so easy for a designer to extract performance aspect information. In this extension, the UML note notation is stilled used to denote additional information associated with that element, such as performance concepts (stereotypes) and their performance values, OCL constraints etc. However, for those performance aspect elements, the triangular symbol is used to specifically indicate that the element is a performance aspect element. In this way, designers could only focus on performance aspect elements in an aspect-oriented model. A graphical representation of the UML extension is depicted in Figure 1, where the UML design element could be a UML class, an UML operation etc.

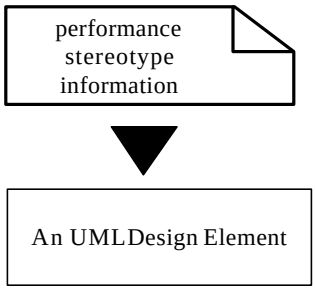


Figure 1. A UML Extension for Modeling Performance Aspect

4. PERFORMANCE ASPECT ANALYSIS

Three techniques suggested by [10] for performance evaluation are analytical modeling, simulation, and measurement. A number of considerations of how to select a technique are also given by it. In current iteration, we consider to use the simulation technique to evaluate response time for the DNS query processing subsystem by using Rapide's analyzing tools. The reason is that simulations can incorporate more details and requires less assumptions than analytical modeling [10].

4.1 Rapide Architecture Description Language

As the UML is a pure modeling language but not a simulation language, we decide to translate the extended UML model into Rapide [16]. As discussed before, Rapide, which has evolved from VHDL, ML, and TSL etc., is an architecture description language for defining and executing models of system architecture. The result of executing a Rapide model is a set of events that occurred during the execution together with casual and timing relationship between events.

Rapide provides five types of sublanguages: the Types language to provide the basic feature for defining interface types and function

types; the Architecture language to extend the Types language with constructs for building interface connection architectures; the Executable Module language to add modules, control constructs, standard types and functions; the Constraint language provides features for expressing constraints on the poset behaviors of modules and functions; and the Event Pattern language to serve as a fundamental part of all of the executable constructs in the executable module, architecture language, and constraint language.

Rapide is presently supported by three kinds of tools for analyzing simulations: constraint checks, to analyzes the conformance of the simulation to the formal constraints defined by the program; poset browsers, to represents graphically the events generated by the simulation, and allows complex manipulation and filtering of the events; animation tools, to depict the execution in a graphical, real-time animation environment.

4.2 Translate UML diagrams into Rapide ADL

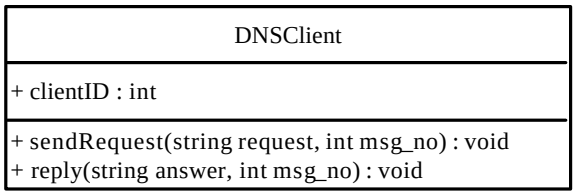
There are several types of UML diagrams. In our approach, extended UML class diagram, statechart diagram, and sequence diagrams/collaboration diagram are selected as the base diagrams to translate. The study of Rapide language has showed that information captured by these diagrams is sufficient for a Rapide architecture simulation:

- UML class diagram is used to describe the static structure of a system and can be translated into the Rapide's Types language. One UML class is translated into one Rapide type interface;
- UML statechart diagram is used to describe the dynamic behavior of an object/system in response to external stimuli and is similar to state transitions described in the behavior part of a Rapide type specification;
- UML sequence diagram/collaboration diagram is used to describe interactions between objects. Information presented in either one of these two diagrams can be used to decide connections between Rapide types as well as events generating order in Rapide;

In our DNS example, which is explained in the Section 5, we extended the conventional UML collaboration diagram with performance aspect information. The performance aspect selected to model and analyze is response time. This performance aspect is translated into Rapide's timing clauses. A Rapide timing clause determines the start and finish time of an event. Thus it can be used to calculate the duration of an event. A detailed description and an example of the translation are presented.

4.3 A Translation Algorithm

To build a tool support for automatic translation of part of UML into Rapide, we have defined and documented the translation algorithms. A simple comparison of a UML class and its Rapide specification is presented in Figure 2. The upper part of this diagram is a simple UML class DNSClient with one public attribute and two operations. The Rapide specification for this class is presented below it. Bold words in this part of specification are Rapide keywords. The translation of class attributes is straightforward. However, to translate UML operations, additional information is needed. As in Rapide, there are two kinds of actions: out actions (declare the types of events of the component may generate and thereby send to other components) and in actions (are used by other components to send events of the action into the component), which are similar with the concepts of stimulus generation events and stimulus reception events discussed in [20].



type DNSClient **is interface**
clientID : **integer**;
action out sendRequest(request : **string**; msg_no : **integer**);
action in reply(answer : **string**; msg_no : **integer**);
end DNSClient

Figure 2. A UML Class and Its Rapide Specification

Algorithm of Translating UML Class into Rapide

Input: a UML class

Output: a Rapide type interface specification

Assumptions:

1. The designer extends the translating UML class's operations with additional modifiers. The first two modifiers are stereotypes <<CRasynch>> and <<CRsynch>> defined in the real time UML [20], where <<CRasynch>> represents the concept of an asynchronous invocation and <<CRsynch>> represents the concept of a synchronous invocation. For a <<CRasynch>> operation, the designers could then use stereotypes <<SGE>> and <<SRE>> further to describe it. <<SGE>> and <<SRE>> are defined in this approach to translate asynchronous invoking/invoked UML operations. The idea behind these two stereotypes is from the real time UML. <<SGE>> stands for stimulus generation event, which is an occurrence of an event that results in the generation of a stimulus. <<SRE>> stands for stimulus reception event, which is an occurrence of an event that represents the acceptance of a stimulus by a receiver instance.
2. <<SGE>> and <<SRE>> have to be used on only public or protected <<CRasynch>> operations. They are not applied to private <<CRasynch>> operations and all <<CRsynch>> operations;
3. <<Cyasynch>> operations should have a void return type;

Algorithm:

1. Output the UML class's name as the name of the Rapide type;
2. Output the UML class's attributes and their types (e.g., integer, string) as the same name attributes and equivalent types for the Rapide type interface;
3. If the visibility of the translating UML attribute is "public' or "protected", the attribute should be translated to a

constituent of the Rapide type's public interface; if the visibility of the translating UML attribute is "private", the attribute should be translated to a constituent of the Rapide type's private interface;

4. If the translating UML operation is a <<CRsynch>> operation, it is translated into a Rapide function with the same name, parameter list, and return type;
5. If the translating UML operation is a <<CRasynch>> one, it is translated into a Rapide action with the same name and same parameter list. If it is also denoted with <<SGE>>, the operation should be translated into a "out" Rapide action; if it is denoted with <<SRE>>, it should be translated into a "in" Rapide action; if it isn't denoted with either one of <<SGE>> and <<SRE>>, it should be translated into a private action for that Rapide type;
6. The translation of UML <<CRsynch>> operations' visibility follows the same procedure of UML attributes' visibility as described in step 3; the translation of UML <<CRasynch>> operations are done by step 5.

5. ILLUSTRATION USING DNS EXAMPLE

An example is given through the Domain Name System server in this section to illustrate the application of the UML extension.

5.1 Domain Name System

The Domain Name System (DNS) [17] provides a way to map a numeric IP address to a character one. IP addresses uniquely identify every computer on the Internet. However, remembering 32 bits numeric address is hard. Therefore, the purpose of DNS is to make it easier for users to access and remember the names of hosts on the Internet. DNS allows networks and hosts to be addressed using common-language names as well as IP addresses and maps host names to various types of addresses through a distributed database.

An example of its use is a simple Internet operation---a hypertext page transfer:

1. A Web browser requested this URL: http://www.FreeSoft.org/Connected/index.html;
2. The DNS protocol was used to convert www.FreeSoft.org into the 32-bit IP address 205.177.42.129;
3. The HTTP protocol was used to construct a GET /Connected/index.html message;
4. A table lookup in /etc/services revealed that HTTP uses TCP port 80;
5. The TCP protocol was used to open a connection to 205.177.42.129, port 80, and transmit the GET /Connected/index.html message;
6. The IP protocol was used to transmit the TCP packets to 205.177.42.129;
7. Some media-dependent protocols were used to actually transmit the IP packets across the physical network.

5.2 Domain Name Server Performance Aspect Modeling Example

The DNS performance aspect modeling example is presented in Figure 3. The main task of a DNS server is to resolve requests from DNS clients. Therefore, query processing is identified as the key performance scenario here. Several processes involved in this scenario are:

- *DNSclient* to send out requests;
- *MessageReceiver* to receive requests;
- *MessageDecoder* to interpret requests;
- *RRRequestProcessor* to search the correct resource records in the server's domain name space;
- *MessageEncoder* to format resource records into a DNS message;
- *MessageSender* to send the answer to the client;

In this example, the response time is the part of the performance aspect of interest. Response time is defined as the interval between a user's request and the system response.
Performance analysis stereotypes from [20] used in this model are:

- *<<PAcontext>>*--models a performance analysis context. This stereotype could associate with a collaboration diagram and has no tags defined;
- *<<PAclosedLoad>>*--models a closed workload (has a fixed number of active or potential jobs). It has four tags already defined:

  PAresTime (response time)

  PApriority (priority)

  PApopulation (population) and

  PAextDelay (external delay).

  We considered that many times designers may want to specify the arrival rate of jobs directly. Therefore, we added an extra tag, *PAarvlRate*, for this stereotype;
- *<<PAstep>>*--models a step in a performance analysis scenario. Tags of this stereotype include:

  PAdemand (host execution demand)

  PArespTime (response time)

  PAprob (probability),

  PArep (repetition),

  PAdelay (delay),

  PAextOp (operations), and

  *PAinterna*l (interval).

PAdemand and *PAdelay* are two tags of the stereotype *<<PAstep>>* and used in our example (Figure 3). *PAdemand* is used to represent the demand execution time of a process, while *PAdelay* is used to represent the time of a delay, e.g., a message delay. Both of them are involved in the calculation of response time. Each tag has been assigned a performance value which is

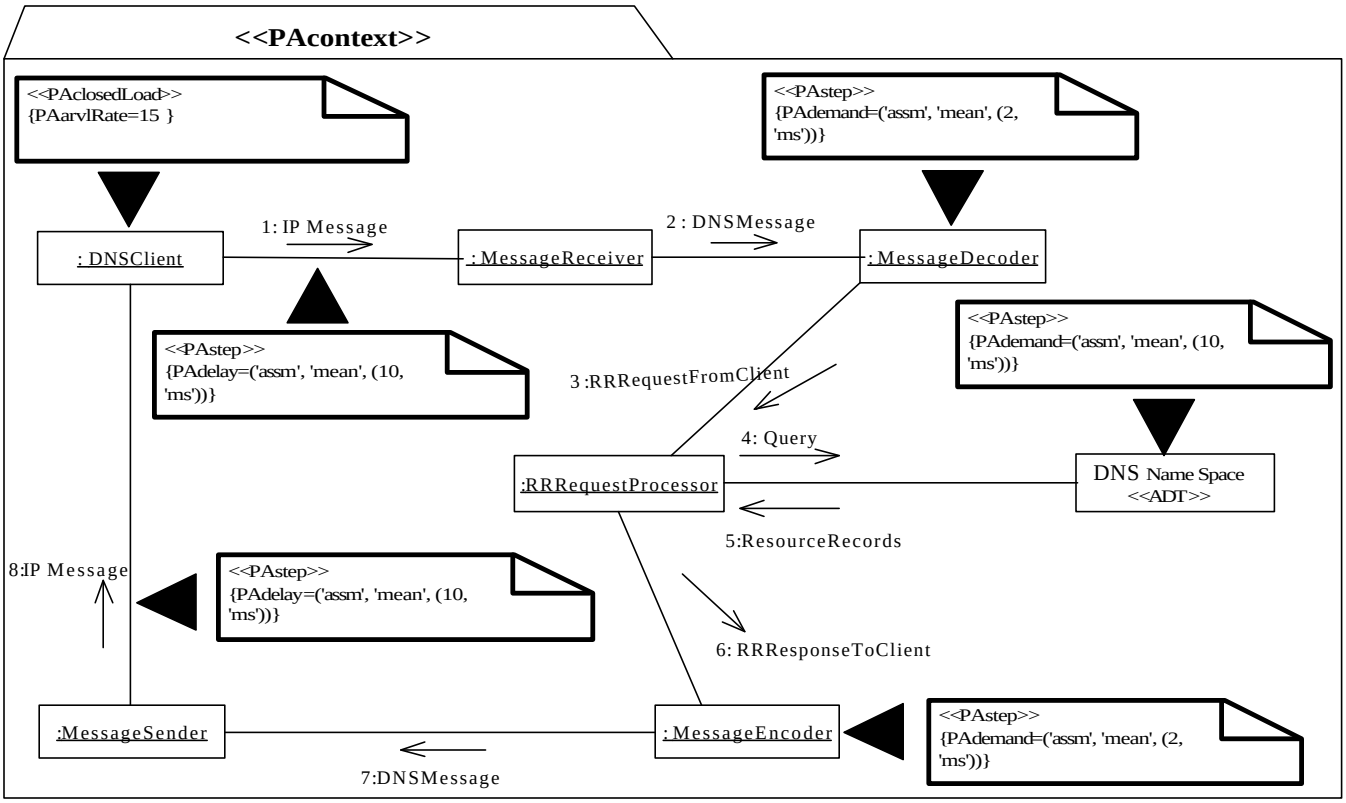


Figure 3. A DNS Performance Aspect Modeling Example

contained in the UML note notation. To obtain an accurate performance evaluation, performance values should be acquired through the process of SPE data gathering [25]. In this example, performance values are estimated. We assume that it takes 10 milliseconds for a message to travel on the network and is expressed as "*PAdelay= ('assm', 'mean', (10, 'ms'))*". The execution time of *MessageReceiver* and *MessageSender* is ignored here as their tasks are just receiving and sending messages. The execution time of *MessageDecoder* and *MessageEncoder* are estimated as 2 milliseconds and is captured in the statement *"PAdemand=('assm', 'mean', (2, 'ms'))*". Same tag is used on *RRRequestProcessor* and we assume that it would take average 10 milliseconds to find the correct answer in the server's domain name space for the request. In these tag expressions, '*assm*' is one of the source modifiers defined in [20] and means that the value is assumed, while 'mean' is one of the type modifiers and means that the value is an average one.

Currently, we manually translate this extended UML collaboration diagram and the corresponding class diagram into Rapide specification (presented in Figure 4) using the algorithms defined. Automatic translation is planned in the next step of our work.

In the Rapide specification, a simplified DNS system is described, which has only one DNS client and one DNS server. In this example, the server only performs query processing activity. On the client side, the object *DNSClient*, for example, is translated as a *DNSClient* interface type in Rapide. A Rapide module called *newDNSClient* is used to specify the client's behavior. On the server side, UML objects are translated into several processes for the *DNSServer* type in Rapide. We simulated that each of these processes (e.g., *MessgeReceiver*, *MessageEncoder*, etc.) executes concurrently and independently of other processes (using Rapide keywords "parallel").

Response time performance aspect information such as the request's arrival rate and processors' execution time in this example are also translated into Rapide's timing clauses. As mentioned in the Section 4.2, a Rapide timing clause determines the start and finish time of an event. Here we use the "pause" timing clause. "Pause" is used after an action call and is followed by an integer, which is called Tick type in Rapide. Ticks are time or values of a clock's counter. Statement *"sendRequest("A DNS request2", 2) pause 4"* means that if the call begins at t Ticks then it will be completed at t+4 Ticks. Another way to interpret this statement is: the DNS client sends a request after every 4 Ticks. One tick can be viewed as a millisecond here since it is a logic clock counter. Therefore, the "pause" timing clause captures those timing information specified in Figure 3.

5.3 Domain Name Server Performance Aspect Analysis Results

In this section we present the analysis results for the Rapide specification presented in Figure 4. Basically, defining, compiling and running a model in Rapide results in a *poset*, which can be represented as a directed acyclic graph. Part of simulation results support for the DNS example is presented in Figure 5 and Figure 6. Data we used in this simulation are: the request's arrival rate is one per 4 Ticks and the DNS server's service time for one

```
type DNSClient is interface
  action out sendRequest(question : data; msg : msg_no);
  action in reply(answer : data; msg : msg_no);
end DNSClient;

type DNSServer is interface
  action out messageSender(answer : data; msg : msg_no);
  action in messageReceiver(question : data; msg :
            msg_no);
end DNSServer;

module newDNSClient() return DNSClient is
parallel
    sendRequest("a DNS request1", 1);
    sendRequest("a DNS request2", 2)  pause 4;
    sendRequest("a DNS request3", 3)  pause 4;
    sendRequest("a DNS request4", 4)  pause 4;
    sendRequest("a DNS request5", 5)  pause 4;
end;
```

```
module newDNSServer() return DNSServer is
  action  messageDecoder(dnsQuestion : data; msg :
            msg_no),
          rrRequestProcessor(dnsQuestion : data; msg :
            msg_no),
          messageEncoder(dnsQuestion : data; msg :
            msg_no);
parallel
  when (?d : data; ?n : integer) messageReceiver(?d, ?n)
  do messageDecoder(?d, ?n) pause 2; end;
  ||
  when (?d : data; ?n : integer) messageDecoder(?d, ?n)
  do rrRequestProcessor(?d, ?n) pause 10; end;
  ||
  when (?d : data; ?n : integer) rrRequestProcessor(?d, ?n)
  do messageEncoder(?d, ?n) pause 2; end;
  ||
  when (?d : data; ?n : integer) m essageEncoder(?d, ?n)
  do messageSender("A DNS answer", ?n); end;
end;
```

Figure 4. DNS Server Querying Processing Subsystem Rapide Specification

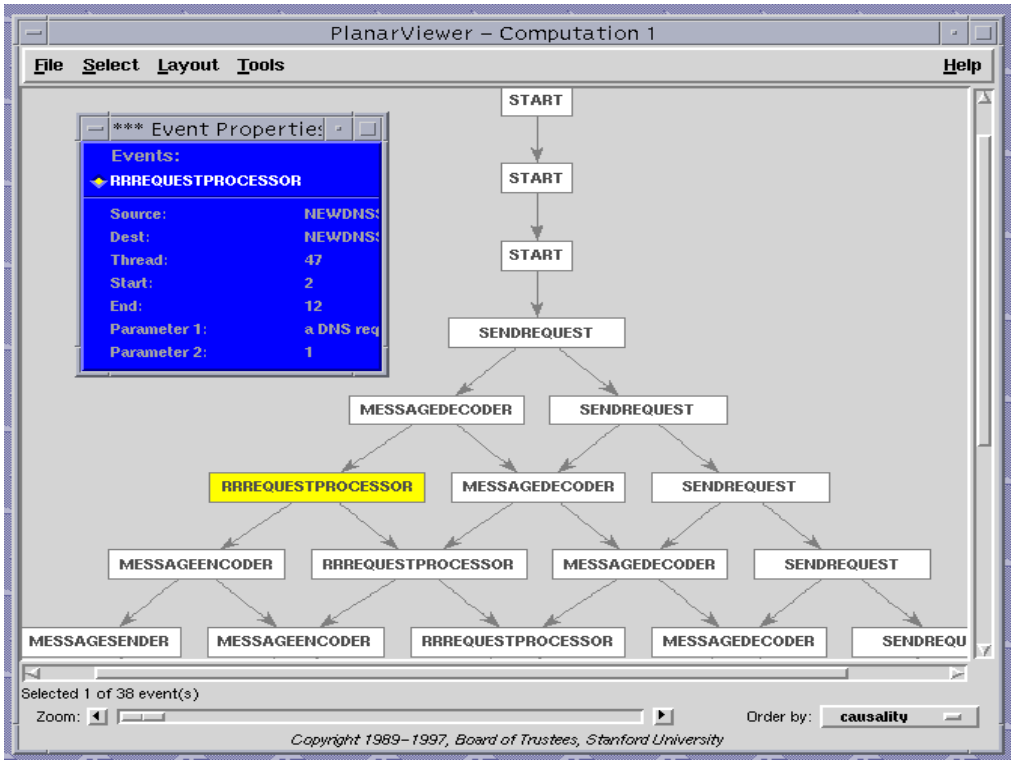


Figure 5. Partial Ordered Event Sets Generated by the DNS

request is assumed as 14 Ticks. Message network delay is ignored here as it is not the critical consideration of a design. Figure 5 presents the *poset* generated by the DNS subsystem simulation. As the whole diagram is very big, only part of the figure is presented here. The diagram is oriented vertically with the causally earlier events at the top. Events generated in the simulation are represented by rectangles and the arrow between two events, for example A $\rightarrow$ B, means that event A happens before event B. Once an event is selected, a popup window is appeared with the information about that particular event, such as its thread ID, timing, parameters etc.

Events' timestamps are presented in Figure 6, where an event's generating thread name, thread number, its start timestamp and

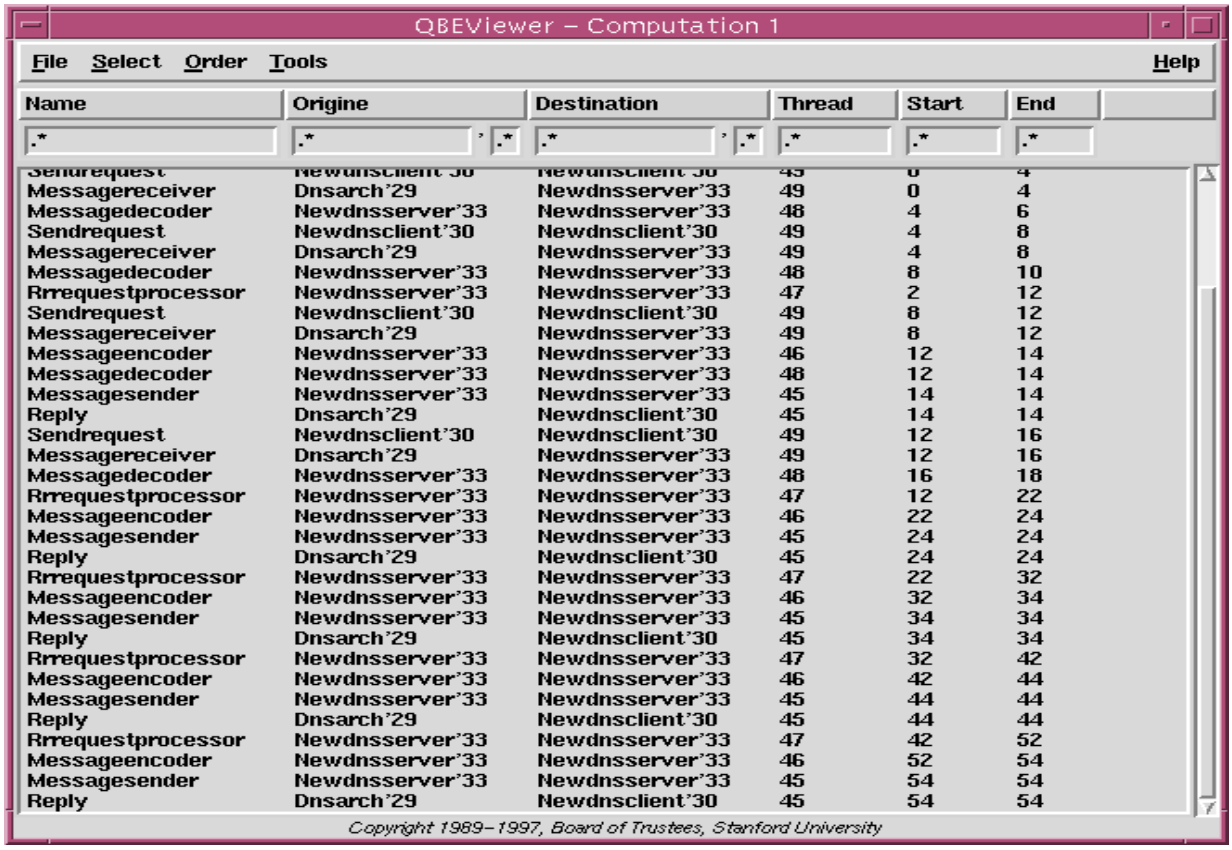


Figure 6. DNS Query Processing Subsystem Event Information

end timestamp are listed in each row. From these timestamps, we can analyze the server's mean response time. In each request sent by the DNS client, we associated a message number with it. In this way, we can retrieve each request's sending time and replying time. Totally, 5 requests are simulated. After calculate the response time for each request respectively, we obtained the analysis result for the server's average response time in this design is 26 Ticks without the consideration of network delay.

In order to clearly illustrate how to model response time performance aspect in an UML extension and then analyze it by using existing analysis tools, we only used one of the DNS subsystems, the query processing subsystem. The activity performed by this subsystem is to read resource records from the server's database. As basically only one kind of activities is involved, the generated *poset* is quite simple and no obvious defects could be detected in the design. However, another important functionality of the DNS server is its periodical zone refreshing, which updates records in the database. Once this part of functionality is added into the design, the Rapide simulation results could be much interesting and valuable since the database has to be consistent all the time. In Rapide, designers can specify these concerns, such as reading and writing the same record at the same time should never happen, race conditions, etc. as constraints of the system. If in the simulation such constraints are violated, a predefined Rapide event "Inconsistent" will be generated and presented in the *poset*. Therefore, the use of the *poset* can help designers to understand the model's behavior, verify the correctness of the model by checking its behavior and any possible constraint violations. Another use of the *poset* is its timing information makes it possible for performance analysis. In our example, the server's response time performance data is obtained. Other performance data can also be obtained include the speedup of a model, cache performance etc. In an early design phase, obtaining these performance data are very important. They can help designers to make decisions among various alternatives, perform trade-off analysis, and select the most suitable one for the system. A research using Rapide to prototype a shared memory multi-processor system, simulate the system and measure performance for three design alternatives can be found in [24].

6. CONCLUSIONS

This paper presents a UML based approach to model and analysis performance aspect. A performance aspect is a set of aspects including response time, rate throughput, resource utilization, probability, time between errors, durations of event, time between events. Response time aspect is defined in extended real-time UML in this approach.

In this approach, the semi-formal extended UML aspect-oriented design model is translated into Rapide ADL to evaluate the system's response time performance aspect. We have used part of the DNS example to illustrate the approach. Part of translation algorithms and Rapide performance simulation results are also presented. The example results prove that response time

performance aspect can be designed and analyzed even in an early stage.

The approach provides a systematic way for designers to model and analysis interested performance aspect in the design phase. One benefit of the approach is it can help designers to evaluate performance for different design alternatives, make decisions among those alternatives, and ensure the final system's completeness and consistency.

We plan to continue this work in a number of interesting directions. One direction is to define and validate additional performance aspects for a complete Domain Name System. We also aim to use other performance evaluation techniques, such as measurement. To use this technique, we selected Armani [18] to translate the UML extension. Armani is a language for capturing software architecture design expertise and specifying software architecture designs. It provides core language constructs to support design analysis. Depending on the estimated data (such as request arrival rate, service time) provided by the designer, Armani's analysis tool can evaluate the design's performance results, e.g., server utilization rate, overloaded component etc. through performance measure techniques such as presented in [10] and [25]. In addition, the algorithms defined to translate the extended UML design into Rapide are going to be captured in tool support, allowing the automatic translation of an extended UML diagram into Rapide.

7. REFERENCES

[1] Aldawud, O., Elrad, T., and Bader, A., "UML profile for aspect-oriented software development", Proceedings of Third International Workshop on Aspect-Oriented Modeling, March 2003.

[2] Balsamo, S., and Simeoni, M., "On transforming UML models into performance models", Workshop on Transformations in UML (ETAPS01), 2001.

[3] Basch, M. and Sanchez, A., "Incorporating aspects into the UML", Proceedings of Third International Workshop on Aspect-Oriented Modeling, March 2003.

[4] Bernardi, S., Donatelli, S., and Merseguer, J., "From UML sequence diagrams and statecharts to analysable petrinet models". Proceedings of the Third International Workshop on Software and Performance (WOSP'2002), July 2002, pp. 35-45.

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

[6] Clements, P.C., "Coming attractions in software architecture", Technical Report No. CMU/SEI-96-TR-008, Software Engineering Institute, Carnegie Mellon University, January 1996.

[7] Cooper, K., Dai, L., Deng, Y. and Dong, J., "Process Definition for the Formal Design Analysis Framework: Creating an Aspect-oriented Design Supporting Response Time Performance", Technical Report UTDCS-20-03, The University of Texas at Dallas, 2003.

[8] France, R., Georg G., and Ray, I., "Supporting multi-dimensional separation of design concerns", Proceedings of the Third International Workshop on Aspect-Oriented Modeling, March 2003.

[9] Hoeben, F., "Using UML models for performance calculation", Proceedings of the Second International Workshop on Software and Performance (WOSP'2000), Sep. 2000, pp. 77-82.

[10] Jain, R., The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley- Interscience, New York, NY, April 1991.

[11] Kazman, R., Abowd, G., Bass, L., and Clements, P., "Scenario-based analysis of software architecture", IEEE Software, November 1996, Vol. 13, Issue: 6, pp. 47-55.

[12] Kazman, R., Barbacci, M., Klein, M., S. Carrière, J., and Woods, S. G., "Experience with performing architecture tradeoff analysis", Proceedings of the 21st International Conference on Software Engineering, May 1999.

[13] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, J., "The architecture tradeoff analysis method**",** Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '98), Aug 1998, pp. 68-78.

[14] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W., "Getting started with AspectJ", Communications of the ACM, October,2001, Vol. 44, Issue: 10, pp. 59-65.

[15] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.M., and Irwin, J. "Aspect-oriented programming", Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97), pp. 220 – 242.

[16] Luckham, D.C., Kenney, J.J., Augustin, L.M., Vera, J., Bryan, D., and Mann, W., "Specification and analysis of system architecture using Rapide**",** IEEE Transactions on Software Engineering, April 1995, Vol. 21, Issue: 4 , pp. 336 -354.

[17] Mockapetris, P.V., "Domain Names - Concepts and Facilities", IETF STD0013, November 1987.

[18] Monroe, R.T., "Capturing software architecture design expertise with Armani", Technical Report No. CMU-CS-98-163, Carnegie Mellon University School of Computer Science, October 1998.

[19] Netinant, P., Constantinides, C.A., Elrad, T., Fayad, M.E., and Bader, A., "Supporting the design of adaptable operating systems using aspect-oriented frameworks", Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), 2000, Vol.1, pp. 271 – 277.

[20] Object Management Group, UML™ Profile for Schedulability, Performance, and Time Specification, OMG Documents ptc/2003-03-02, March 2003.

[21] Pawlak, R., Duchien, L., Florin, G., Legond-Aubry, F., Seinturier, L., and Martelli, L., "A UML notation for aspect-oriented software design", International Workshop on Aspect-Oriented Modeling with UML, April 2002.

[22] Pinto, M., Fuentes, L., Fayad, M., and Troya, J.M., "Separation of coordination in a dynamic aspect oriented framework", Proceedings of the First International Conference on Aspect-oriented Software Development, April 2002, pp. 134 – 140.

[23] Pooley, R., "Using UML to derive stochastic process algebra models", Proceedings of XV UK Performance Engineering Workshop, 1999.

[24] Santoro, A., "Case study in prototyping with Rapide: a shared memory multiprocessor system", Technical Report CSL-TR-93-564, Computer Systems Laboratory, Stanford University, March 1993.

[25] Smith, C. U., Performance Engineering of Software Systems, Reading, MA, Addison-Wesley, 1990.

[26] Smith, C.U., and Williams, L.G., "Performance evaluation of software architectures", Proceedings of the First International Workshop on Software and Performance (WOSP'1998), 1998, pp. 164-177.

[27] Spitznagel, B., Garlan, D., "Architecture-based performance analysis", Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering (SEKE'98), June 1998.

[28] Williams, L. G., and Smith, C. U., "PASA^{SM}: a method for the performance assessment of software architectures", Proceedings of the Third international Workshop on Software and Performance (WOSP'2002), July 2002, pp 179-188.