Process Definition for the Formal Design Analysis Framework

Creating an Aspect-oriented Design Supporting Response Time Performance

Technical Report UTDCS-20-03

K. Cooper

Lirong Dai

Jing Dong

Department of Computer Science
Mail Station 31
Univ. of Texas at Dallas
Richardson, TX, USA P.O. 830688

Table of Contents

1. Introduction 1
  1.1 Motivation and Problem Description 1
  1.2 Report Organization 1
2. Related Work 2
  2.1 Aspect-Oriented Approach 2
    2.1.1 Aspect-Oriented Software Architecture 2
    2.1.2 Dynamic Aspect-Oriented middleware Framework 2
    2.1.3 UML All pUrpose Transformer 3
  2.2 Architecture Description Languages 3
    2.2.1 WRIGHT 4
    2.2.2 RAPIDE 4
    2.2.3 DARWIN 5
    2.2.4 ACME 5
  2.3 UML 6
3. Overview of the Formal Design Analysis Framework 8
4. Formal Design Analysis Framework Process Model 11
  4.1 Activity 1. Create Semi-formal Extended UML Aspect-oriented Design Model 11
  4.2 Activity 2. Create Formal Aspect-oriented Models 12
  4.3 Activity 3. Evaluate the Quality of the Semi-formal Aspect-oriented Model 13
5. Illustration: Create a Semi-formal Extended UML Aspect-oriented Design Model 14
  5.1 Base Models of the DNS 14
    5.1.1 DNS Use Case Model 15
    5.1.2 DNS Architecture Design Model 15
    5.1.3 DNS Class Diagram Models 17
    5.1.4 DNS Query Processing Sequence Diagram 19
  5.2 Adding a Performance Aspect 21
    5.2.1 Performance Aspect Modeling 23
    5.2.2 Performance Aspect Analysis 24
6. Conclusions and future work 27
7. References 29
Appendix A. Performance Aspect 32
Appendix B. Security Aspect 33

Abstract

Much attention has recently been focused on the problem of effectively developing software systems that meet their non-functional requirements. Architectural frameworks have been proposed as a solution to support the design and analysis of non-functional requirements such as performance, security, adaptability, etc. A significant benefit of performing such analysis work is to detect and remove defects earlier in the design phase. However, the non-functional properties of a system are mostly addressed independently making it difficult for the design and analysis of a system with respect to a collection of these interacting, or tangled, properties. This report presents an overview of our proposed solution to this problem called the Formal Design Analysis Framework (FDAF). More specifically we describe the process that supports the definition of an aspect-oriented design. The process is described using activity diagrams. An example is given to model part of the design for a Domain Name System (DNS) for one aspect, or non-functional properties, performance. The unified modeling language (UML) and the architectural description language Rapide and Armani are used in the example. The long-term goal of the work is to support the definition and analysis of two or more synergistic or conflicting aspects, or non-functional requirements, using the non-functional requirement (NFR) Framework.

1. Introduction

1.1 Motivation and Problem Description

Attention has recently been focused on the problem of effectively designing software systems that meet their non-functional requirements. The motivation for such research is straightforward: one of the significant, expected benefits of performing such analysis work is to detect and remove defects earlier in the design phase resulting in a higher quality system, reduced development time, and reduced development costs [43].

Researchers have proposed solutions to this problem from a number of different directions. Architectural frameworks have been proposed as solutions to support the analysis of a single non-functional property such as security [18], performance [45], or adaptability [7], or provide a reasoning framework [24]. The unified modeling language (UML) has been investigated with respect to how it can be used and extended to model non-functional properties in a design. Aspect-oriented design approaches have been proposed as a means to systematically model cross cutting concerns like non-functional properties.

The non-functional properties, or aspects, of a system are mostly addressed independently making it difficult for the design and analysis of a system with respect to a collection of these properties. Current approaches either concentrate on serving as a general-purpose architecture modeling language within a particular domain, or support the analysis of one specific non-functional requirement of a system (e.g., security, performance, adaptability, etc.). To fill this gap in the research, we ask the following question:

> *Can we define an aspect-oriented architectural framework that supports the design and analysis of multiple, non-functional properties for distributed, real-time systems?*

As a solution, we propose the Formal Design Analysis Framework (FDAF). Our vision is that FDAF is intended to support an aspect-oriented design and analysis of multiple, non-functional properties using the Non-Functional Requirement (NFR) Framework, UML, and a set of formal methods. The NFR Framework is used to identify, describe, prioritize, and negotiate the relationships among non-functional goals (called softgoals) of a system [8]. UML is selected because it is a well-known, standardized notation. Extensions to UML are proposed to better support the description of non-functional properties. A set of formal methods is used because a single formal method is not available that is well suited for defining and analyzing numerous non-functional properties for a system (performance, security, capacity, etc.). Existing formal methods are used because developing a new formal method is a substantial task; we have chosen to leverage the previous work in architectural description languages (ADL), Petri Nets (timed, stochastic), temporal logic, etc. and integrate them into our framework rather than develop a new formal notation and tool support.

This report is focuses on the first step of the FDAF, which is the creating of semi-formal extended UML model to support aspect-oriented design. The UML extension is illustrated by a Domain Name Server (DNS) example.

1.2 Report Organization

This report is organized as follows. Related work is presented in Section 2. An overview of FDAF is presented in Section 3. The description of our process is in Section 4. Part of the process is illustrated in Section 5. Conclusions and Future work are in Section 6.

2. Related Work

As our work draws upon aspect-oriented approaches and the extension and translation of UML into the ADL RAPIDE, we provide a brief survey of work in these areas.

2.1 Aspect-Oriented Approach

Aspect-oriented approaches [38] rely on the principle of separation of concerns. A property of a system needs to be implemented as an aspect when it cannot be clearly encapsulated in a generalized procedure. Examples of aspects include non-functional properties such as performance, security, adaptability, availability, etc. Aspect-oriented design has been inspired by the research in aspect-oriented programming (AOP) [13]. This is a relatively new programming technique that is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and their relationships. AOP relies on mechanisms in the underlying environment to weave, or compose, a coherent system [13]. A survey of aspect-oriented architectural frameworks is presented in this section.

2.1.1 Aspect-Oriented Software Architecture

Aspect-Oriented Software Architecture (AOSA) [36] is developed to support designers and programmers in cleanly separating components and aspects from each other in different layers. AOSA uses Aspect-Oriented Frameworks [34]. In this approach, aspects are defined as properties of the system that tend to cut across groups of functional components. Those aspects but do not necessarily align with the system's functional components, but they increase interdependencies between components and thus affect the quality of a system.

AOSA is based on decomposition of aspects in system design that consists of components, aspects, and layers:

- Component---Components consist of the basic functionality modules of the system such as the file system, communication, and process management etc.;
- Aspect---Aspects are crosscutting entities, and they include fault tolerance, synchronization, scheduling, naming etc.;
- Layer---Layers consist of the components and aspects decomposed into a number of more manageable sub-problems.

Separating components, aspects, and layers from each other made it possible to abstract and compose them to produce the overall system. AOSA achieved reusability and stability by the higher level can use the lower level of the implementation and the high level would not be aware of if there was a change in the low lever.

2.1.2 Dynamic Aspect-Oriented middleware Framework

Dynamic Aspect-Oriented middleware Framework (DAOF) is proposed by [41]. In DAOF, software components and aspects are first-order entities composed dynamically at run-time according to architectural information stored in a middleware layer. Java is used as the general-purpose language to implement both components and aspects in this approach. The approach aims to providing a basis to separate any kind of aspects. Collaborative Virtual Environment (CVE) has been chosen to be the application domain.

The framework is composed of application architecture (AA), DAOF components and aspects, and a middleware layer. The architecture of an application is normally spread all over the system and describes which components set up the system and how they interact to accomplish the required functionality. Information about when and how to apply aspects to components is not hard coded. A unique and universal role name is assigned to name references form components and aspects code so that components and aspects with the same role and provide the same behavior can be replaced by equivalent components or aspects.

The AA of a system is defined as a list of components and aspects that can be instantiated in the system and a set of Architecture Restrictions (ARs), and also each of these components and aspects is defined by a role name, an interface and an implementation class, where interfaces are detached from implementation classes. Aspects that must be created at run-time have these alternatives: environment-oriented, user-oriented, type-oriented and component oriented. This information is just a part of the aspect description inside AA and later the middleware layer creates aspect instance according to this information. Thus the number of types of aspects that are applicable to a component can change dynamically.

2.1.3 UML All pUrpose Transformer

UMLAUT (UML All pUrpose Transformer) [19], a framework for weaving UML-based aspect-oriented designs, is introduced as a tool for "weaving" aspects when modeling with the UML, and as a methodological support for building and manipulating UML models with UML.

UMLAUT's architecture has a three-layer architecture. The input front end consists of a graphical user interface for interactive editing and another interface for reading UML models described in various formats (XMI, Rational Rose^{TM} MDL, Eiffel source, Java source). The middle core engine is made up of the UML meta-model repository and the extensible transformation engine. And the output back end contains various code generators. The meta-model in UMLAUT's core engine is implemented as a set of collaborative Eiffel classes. The resulting implementation is a direct mapping of UML meta-classes of Eiffel classes. The transformation engine of UMLAUT is responsible for the weaving process. A designer specifies the required transformation by composing a set of operators from the UMLAUT library. And users may also add new operators and extend the existing library to support different weaving operations. The framework is designed to cater to three levels of users: model designers, implementation architects and framework implementers and help the designer to programming "weaving" of the aspects at the level of the UML meta-model.

2.2 Architecture Description Languages

Architectural description plays an increasingly important role in the process of describing and understanding software systems. A number of ADLs [30] have been developed as formal notations to represent and reason about software architectures. A survey of ADLs is provided in this section. We recognize this survey is not exhaustive.

2.2.1 WRIGHT

WRIGHT [1] is an architectural description language that provides formal description for both architectural configurations and architecture styles. WRIGHT uses explicit, independent connector types as interaction patterns and describes the abstract behavior of components using a CSP-like notation. It focuses on modeling and analysis of the dynamic behavior of concurrent systems [30]. A collection of static checks is used to determine the consistency and completeness of an architectural specification in WRIGHT.

WRIGHT is built on the basic architectural abstractions of components, connectors, and configurations. Explicit notations for each of these elements have been provided in WRIGHT:

- Component---A component describes a localized, independent computation with two parts of descriptions, the interface and the computation. An interface consists of a number of ports, where each port represents an interaction in which the component may participate. The computation section of a description describes what the component actually does. The computation carries out the interactions described by the ports and shows how they are tied together to form a coherent whole;
- Connector---A connector represents an interaction among a collection of components;
- Configuration---A configuration is a collection of component instances combined via connectors.

In a WRIGHT description, instances of each component and connector type are required to be explicitly and uniquely named if they appear in a configuration. Once the instances have been declared, a configuration is completed by describing its topology. This is done by associating a component's port with a connector's role.

Hierarchical descriptions are supported in WRIGHT by representing an architectural description as the computation of a component, where the component serves as abstraction boundary for a nested architectural subsystem. In addition to describing and analyzing system configurations, WRIGHT permits the designer to define architectural styles. A WRIGHT style has two parts: the common vocabulary and constraints on configurations. A common vocabulary is introduced by declaring a set of component and connector types. In WRIGHT, the notation for constraints is based on first order predicate logic.

2.2.2 RAPIDE

RAPIDE is an event-based concurrent object-oriented language designed for simulation and behavioral analysis of architectures of distributed systems at an early stage [26]. It has been developed:

- To allow be expressed in an executable form for simulation;
- To adopt an execution model which captures distributed behavioral and timing as precisely as possible;
- To provide formal constraints and mappings to support constraint-based definitions for conformance to architectures standards;
- To address the issues of scalability involved in modeling industry system architectures.

An architecture in RAPIDE consists of a set of specifications (called interfaces) of modules, a set of connection rules that define direct communication between the interfaces, and a set of formal constraints that define legal and/or illegal patterns of communication. An interface is a definition of the features provided to the architecture and required from the architecture by modules that conform to the interface. It may contain an abstract definition of the behavioral of modules. Behavioral are defined by executable reactive rules, which specifies relationships between data received and data generated by a module. Connections define either synchronous or asynchronous communication of data between interfaces, also in a simple interactive executable form. Formal constraints specify restrictions on various aspects of interfaces and connections. These architecture constructs have an event-based execution model, called the *POSET* model. Interface behaviors execute by waiting to receive certain sets of events and then reacting by generating new events. Connections define how events generated at some interfaces cause other events to be received at other interfaces. Constraints place restrictions on event activity, both in interfaces and over the set of connections and they are checkable.

RAPIDE consists of five major independent languages:

- types language for describing the interfaces of components
- architecture language for describing the flow of events between components
- specification language for writing abstract constraints on the behavioral of components
- executable language for writing executable modules
- pattern language for describing patterns of events.

2.2.3 DARWIN

DARWIN is a language for describing hierarchic configuration structures [27][28]. Unlike other module-interconnection language, DARWIN addresses the dynamic aspects of configuration as well as providing for static configuration. A DARWIN configuration structure can be viewed as a hierarchy of interconnected component instances. Each level of the hierarchy being described with a separated DARWIN configuration description termed as a composite component type, which is constructed from the primitive computational components and these in turn can be figured into more complex composite types.

In DARWIN, a component is defined by the services it provides to other components and the services it requires from other components. Components interact by accessing services. A DARWIN configuration description includes component instantiation declarations and binding specifications between a required service and a provided service. DARWIN supports the description of dynamically reconfiguring architectures through two constructs---lazy instantiation and explicit dynamic constructions. Using lazy instantiation, a logically infinite configuration is described and components are instantiated only as the services they provide are used by other components. Explicitly dynamic structure is provided through the use of imperative configuration on constructs. The operational semantics of DARWIN in terms of the π-calculus, Milner's calculus of mobile process, is described in [27] and [28].

2.2.4 ACME

ACME [17] is developed as an architecture interchange language with the purpose of providing a common intermediate representation for a wide variety of architecture tools. ACME supports mapping of architectural specification from one ADL to another; it is not strictly an ADL [30].

ACME is built on a core ontology of seven types of entities for architectural representation: components, connectors, systems, ports, roles, representations, and rep-maps.

- Component---A components represents the primary computational elements and data stores of a system.
- Connector---A connector represents interactions among components.

- System---Systems represent configurations of components and connectors.
- Port---Ports are defined as components' interfaces.
- Roles---Roles are defined as connectors' interfaces.
- Representations---Any components or connector can be represented by one or more detailed, low-level descriptions, each such description is termed a representation in ACME.
- Rep-maps---When a component or connector has an architectural representation there must be some way to indicate the correspondence between the internal system representation and the external interface of the components or connector that is being represented, a rep-map defines this correspondence. The topology of this system is declared by listing a set of attachments.

Additionally, the ACME provides an open semantic framework in which architectural structures can be annotated with ADL-specific properties. This open semantic framework allows specific ADLs to associate computational or run-time behavior with architectures using the property construct, and also provides a straightforward mapping of the structural aspects of the language into a logical formalism based on relations and constraints. An ACME specification represents a derived predicate, which can be reasoned about using logic or it can be compared for fidelity with real world artifacts that the specification is intended to describe.

2.3 UML

UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system [20], [40]. UML based frameworks have also been proposed as solutions that support the analysis of non-functional requirements [45]. Significant advantages of using UML include that it is a standardized and well-known notation. In architectural research, the UMLAUT Framework [19] uses standard UML while other research groups focus on formalizing a subset of UML [29], modeling architectures using UML in combination with the object constraint language (OCL) [31] and extending UML [45].

UML provides extension mechanisms to allow the user to model software systems if the current UML technique is not semantically sufficient to express the systems. These extension mechanisms are stereotypes, tagged values, and constraints.

Stereotypes allow the definition of extensions to the UML vocabulary, denoted by *<<stereotype-name>>*. The base class of a stereotype can be different model elements, such as Class, Attribute, and Operation. A *stereotype* groups tagged values and constraints under a meaningful name. When a stereotype is branded to a model element, the semantic meaning of the tagged values and the constraints associated with the stereotype are attached to that model element implicitly. A number of possible uses of stereotypes have been classified in [5].

Tagged values extend model elements with new kinds of properties. Tagged values may be attached to a stereotype, and this association will navigate to the model element to which the stereotype is branded. Basically, the format of a tagged value is a pair of name and an associated value, i.e., {name=value}. Note that the tagged values attached to a stereotype must be compatible with the constraints of the stereotype's base class.

Constraints add new semantic restrictions to a model element. Typically constrains are written in the Object Constraint Language (OCL) [47]. Constraints attached to a stereotype imply that all model elements branded by that stereotype must obey the semantic restrictions which constraints state. Note that the constraints attached to a stereotyped model element must be compatible with the constraints of the stereotype and the base class of the model element.

A *profile* is a stereotyped package that contains model elements that have been customized for a specific domain or purpose by extending the metamodel using stereotypes, tagged values, and constraints. A profile may specify model libraries on which it depends and the metamodel subset that it extends.

Figure 1 shows the relationships among stereotype, constraint, tagged value, and model element, where the stereotype, constraint, and tagged value can apply to a model element, and add corresponding semantics to that model element. Constraints and tagged values can also apply to a stereotype and the corresponding semantics added to the stereotype will navigate to a model element when the stereotype is branded to the model element.

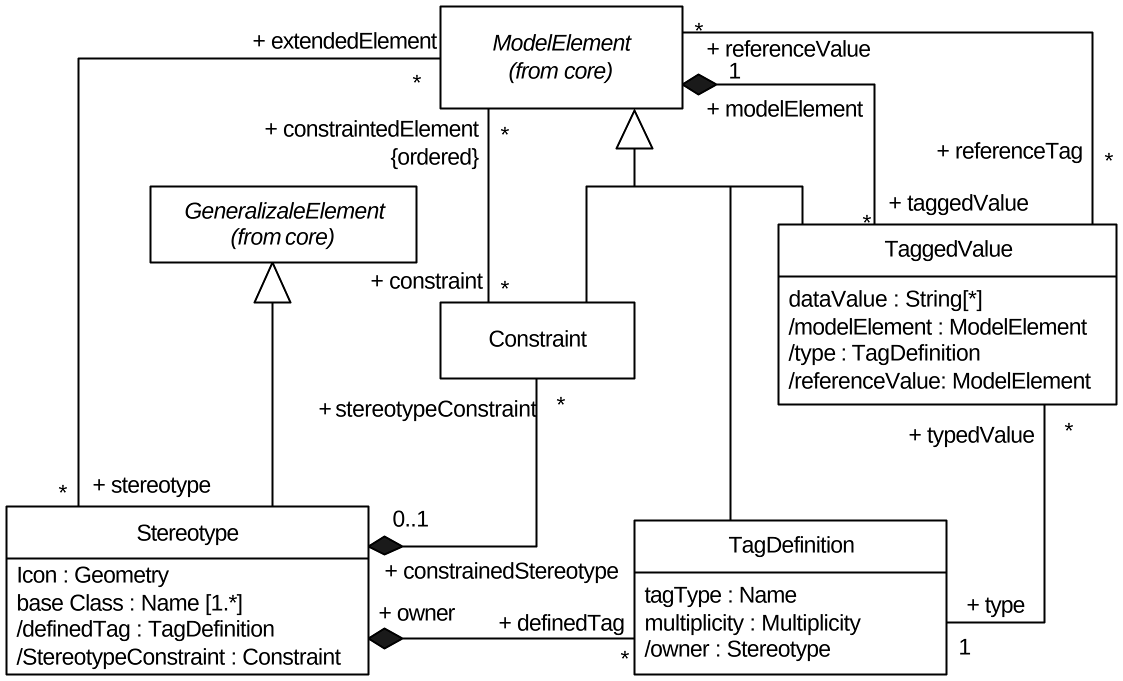


Figure 1. UML Profile Meta Model

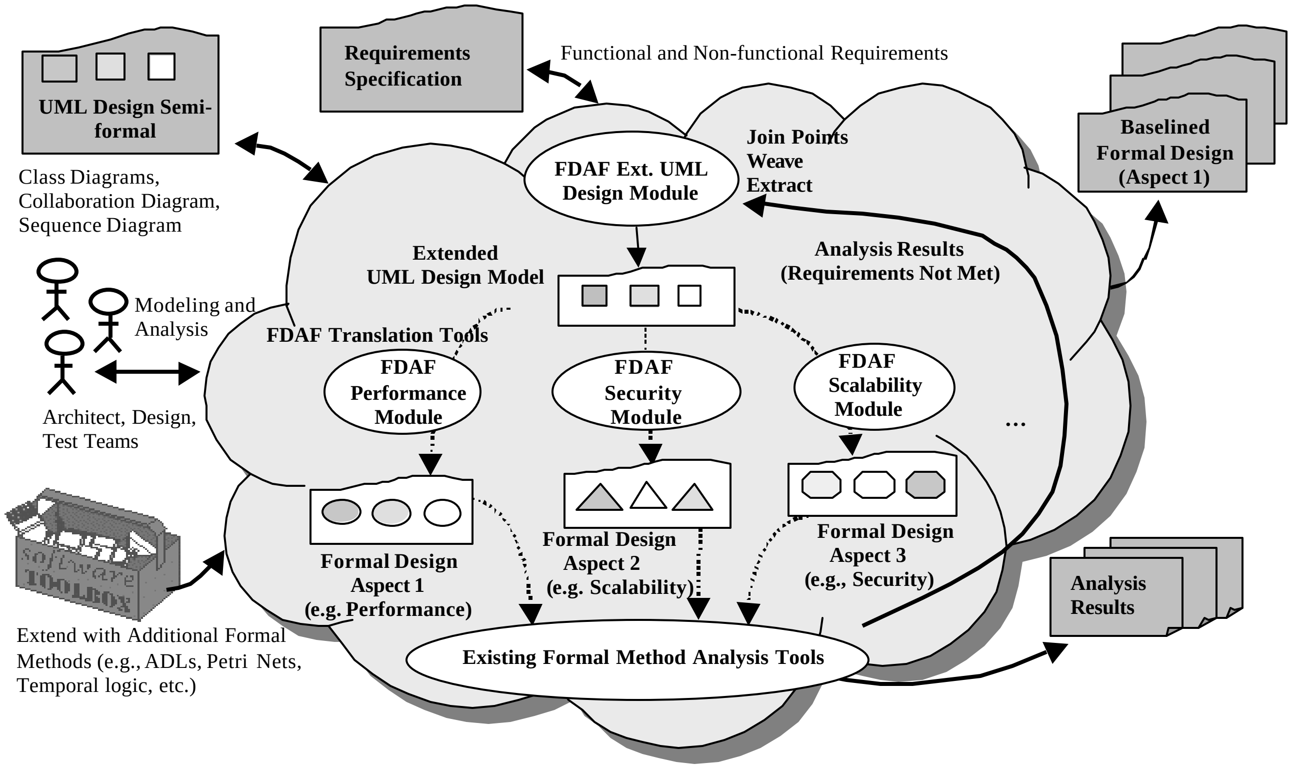
A variety of UML extensions, such as [2] and [42] have been proposed to model non-functional aspects. Approaches focus on dealing with performance can be found in [3], [25] and [32]. In [2], an UML extension is introduced to model aspects with new design elements added into the current existing UML. By applying this approach, the project is assumed to be developed using aspect-oriented programming language AspectJ. [42] presents another UML extension to incorporate non-functional aspects (e.g., performance, reusability, portability etc) through the use of additional stereotypes, class compartments and dependencies. Additional stereotypes defined in this approach are based on the NoFun notation [15] and the OCL is used to establish the constraints of the incorporated non-functional attributes. The NoFun notation is developed to describe non-functional requirements within the component-programming framework, which limits the UML extension only suitable for software component applications.

3. Overview of the Formal Design Analysis Framework

The purpose of the FDAF is to support the systematic design of a system that meets its non-functional requirements such as performance, security, scalability, etc. [9]. The framework has a defined process and product model. An overview of the FDAF is illustrated in Figure 1. In this figure, the FDAF is represented with a cloud surrounded by the stakeholders, inputs, and outputs of the framework. The stakeholders are the designers, requirements engineers, and formal methodologists who use the framework to develop and evaluate a system design that meets its functional and non-functional requirements.

The inputs include an object-oriented design model documented in the UML and a requirements specification that includes the functional and non-functional requirements for the system. The framework may be extended with additional formal methods; we have included a formal methods toolbox on Figure 2 to represent this. UML [20],[40] is selected as the notation for the input model because it is a readable, extendable [11], well-known notation that provides strong support for describing the functional capabilities of a system. The notation, however, has limited support for modeling non-functional properties.

The outputs include a set of aspect-oriented formal design models and the analysis results. In our work, an aspect-oriented design is a solution defined to support one or more non-functional requirements for the system. Within the framework there are components including an extended UML notation and tool support. The tool allows the user to browse and select re-usable aspects, define new aspects, define join points in a design for static and dynamic views of the design, weave an aspect into the design, extract an aspect out of the design, assist the user in selecting a formal method, and translates an extended semi-formal UML design into a formal



Requirements Specification
Functional and Non-functional Requirements
UML Design Semi-formal
Class Diagrams, Collaboration Diagram, Sequence Diagram
Join Points
Weave
Extract
FDAF Ext. UML Design Module
Baselined Formal Design (Aspect 1)
Extended UML Design Model
Analysis Results (Requirements Not Met)
Modeling and Analysis
FDAF Translation Tools
FDAF Performance Module
FDAF Security Module
FDAF Scalability Module
...
Architect, Design, Test Teams
Software TOOLBOX
Formal Design Aspect 1 (e.g. Performance)
Formal Design Aspect 2 (e.g. Scalability)
Formal Design Aspect 3 (e.g., Security)
Analysis Results
Extend with Additional Formal Methods (e.g., ADLs, Petri Nets, Temporal logic, etc.)
Existing Formal Method Analysis Tools

Figure 2. Overview of Formal Design Analysis Framework

notation.

NFR Framework for Design. Fundamental to software engineering is the concept of meeting the needs of a customer. As presented in [7], we need to recognize that in reality a customer is not likely going to be 100% satisfied with a delivered product, the concept of "satisficing" has been introduced [44]. The term *satisficing* means that the customer is satisfied enough with the product to use it (i.e., it is "good enough"). The idea of *satisficing* has been adopted in the NFR Framework [8]. This framework supports the description, analysis and negotiation of multiple, possibly conflicting non-functional properties. An example of conflicting properties is the need for high security and fast performance.

FDAF adopts the concepts of the NFR Framework [8] while developing an extended UML design for a system. The concepts adopted include the identification, analysis, and negotiation of design alternatives. In our work, we extend the NFR Framework to support the design of the system. In this process, non-functional aspects are stated and managed through refining and inter-relating aspects, justifying and documenting decisions, and determining their impacts. Addressing the conflicting aspects earlier in an aspect-oriented design process are expected to reduce the rework needed later. We believe that considering the conflicts and trade-offs between aspects in a final, weaving step may be too late, as the architectures for each aspect, developed independently, may not be possible to weave together. Instead, we consider the conflicts and trade-offs earlier as a concurrent and interleaving step in FDAF.

Use a Repository of Aspects. The framework provides a repository of aspects defined in UML. The aspects are defined using class diagrams, sequence diagrams, collaboration diagrams, or OCL. The designer can search the repository and select parts of the aspects needed in the system design. Examples of aspects are presented in Appendix A.

Use Existing Formal Methods. The framework assists the designer in selecting a formal method for each non-functional aspect of the system. Existing formal notations normally are only suitable for describing one or a few types of system properties. By adopting the aspect concept and a set of formal methods, we can select the most suitable notation and analysis techniques for a given property.

The need to automate the analysis of an architecture (to reduce the cost, reduce the time, and improve the quality of the analysis), leads to us to consider the use of formal methods. ADLs, Petri Nets, and temporal logic have already been proven useful in designing and analyzing specific non-functional aspects of a design; we plan to build on the existing work in the area. Formal methods used to model software architectures include Petri Nets [6], temporal logic [18], process algebra [4], and ADLs including Wright[30], ACME [30], and RAPIDE [26].

Currently, we support RAPIDE in the framework as a formal method to support modeling and analyzing performance aspects of the DNS system. RAPIDE is an event-based, concurrent, object-oriented language designed for the simulation and behavioral analysis of distributed system architectures [26]. RAPIDE adopts an execution model which captures distributed behavioral and timing as precisely as possible. By utilizing architecture definitions as the development framework, RAPIDE allows gradual refinement of architectures into products, and supports testing and maintenance based on automated comparison with formal standard architectures. Meanwhile, tool support for RAPIDE is available and can be used to verify the specification for our example system, the DNS. Additional formal methods are going to be supported as the work matures.

Aspect-oriented Formal Models. Using the principle of separation of concerns, a set of simpler models, each built for a specific purpose (or aspect) of the system, can be defined and analyzed [38]. Each aspect model can be constructed and evolved relatively independently from other aspect models. Since an aspect model focuses on only one type of property, without burden of complexity from other aspects, an aspect model is potentially much simpler and smaller than a traditional mixed system model. This is expected to dramatically reduce the complexity of understanding and analysis.

Automate the Translation from Semi-formal to Formal Notations. An automated translation ensures the consistency between the semi-formal extended UML model and the formal models. Currently, the extended UML is manually translated into RAPIDE. In the future, the extensions in UML are going to be refined such that we can define and implement algorithms to partially automate the translation.

Analyze the Formal Model. Once translated, the formal model can be analyzed for specific properties using its existing tool support (e.g., model checkers, theorem provers, etc.). If the property does not hold in the formal model, then the architects and designers revise and possibly renegotiate the requirements. They need to modify the semi-formal models and update the formal models. The analysis and modifications are performed iteratively until the desired properties hold true. Since NFRs might not be absolutely achieved, they may simply be satisfied sufficiently ("satisficed") [44].

4. Formal Design Analysis Framework Process Model

A process is defined to describe the activities that accomplish the goal of systematically defining an aspect-oriented design that meets the system's functional and non-functional requirements (refer to Figure 3). The activities have entry and exit conditions, inputs, outputs, who performs the activity, and a description of the steps

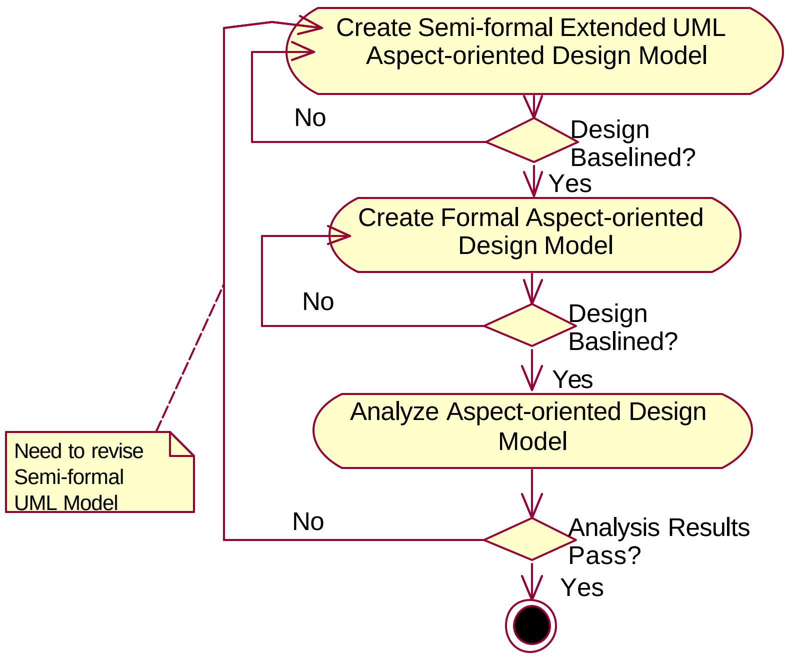

Create Semi-formal Extended UML Aspect-oriented Design Model
No
Design Baselined?
Yes
Create Formal Aspect-oriented Design Model
No
Design Baslined?
Yes
Analyze Aspect-oriented Design Model
Need to revise Semi-formal UML Model
No
Analysis Results Pass?
Yes

Figure 3. Formal Design Analysis Framework Process

in the activity. A symbolic example is used to describe the steps.

4.1 Activity 1. Create Semi-formal Extended UML Aspect-oriented Design Model

Entry Condition: A UML design model that meets the functional requirements of the system is defined. Static (class diagrams) and dynamic (sequence or collaboration diagram) views of the design are available. Although design patterns [16] are likely to have been selected and applied to consider the non-functional requirements of the system, the design may not explicitly describe how the design meets these requirements.

Exit Condition: Extended UML Design Model is Baselined

Who performs activity: Requirements Engineer, Designer

Description:

Step 1.1 The requirements engineer and designer review the UML design model and the non-functional requirements of the system and begin to decompose, describe relationships among, and prioritize the non-functional requirements of the system. The NFR approach is extended and used to accomplish this. Based on the results, the designer selects a related set of non-functional requirements, or aspects, to integrate into the design.

Step 1.2 The designer browses the FDAF repository for pre-defined aspects and selects a candidate that matches the non-functional requirements for the system. Currently, aspects are defined in UML as subsystem designs (i.e., class diagrams, capturing the static view of the aspect, and sequence or collaboration diagrams, capturing the dynamic view) or in OCL. The two kinds of definitions are needed because the aspects may be prescribed properties that permeate all or part of a system (e.g. meet a response time requirement) or as capabilities that may be delivered by developing classes (e.g. provide secure access control). Classes are defined with invariants, pre-conditions, and post-conditions on public methods. If a match is not found, then the designer needs to define the aspect and add it to the repository.

Step 1.3 The designer defines the point cuts points in the dynamic view of the design model. A triangle symbol is used to indicate where in the dynamic model all or part of an aspect needs to be included. The static view is updated such that it is consistent with the dynamic view.

Step 1.4 The designer weaves an aspect into the design. Currently, this step is performed manually. Tool support to automate this step is planned for future work.

These steps are illustrated in Figure 4.

4.2 Activity 2. Create Formal Aspect-oriented Models

Entry Condition: Extended UML Design Model is Baselined

Exit Condition: Formal Aspect-oriented Model is Baselined

Who performs activity: Requirements Engineer, Designer

Description:

Step 2.1 The designer selects a formal notation to use. A decision tree is defined to assist the designer in selecting a formal notation. Questions that are used to select a formal notation include:

Is the system concurrent?

Is the system real-time?

Are there safety or liveness properties of the system that need to be met?

Are there performance, security, adaptability, etc. requirements that need to be met?

What kind of analysis needs to be performed to demonstrate the performance, security, adaptability, etc. requirements are met?

…

The decision tree for the framework is going to be defined as the work progresses. Currently, only the ADL Rapide is supported in the framework. Rapide is suitable for modeling and analyzing concurrent systems with constraints, such as performance constraints. When the FDAF is extended with an additional formal notation, the decision tree is also updated.

Step 2.2 The designer translates the extended semi-formal UML design into a formal notation using the FDAF tool support.

4.3 Activity 3. Evaluate the Quality of the Semi-formal Aspect-oriented Model

Entry Condition: Formal Aspect-oriented Model is Baselined

Exit Condition: Quality of Extended UML Model is verified

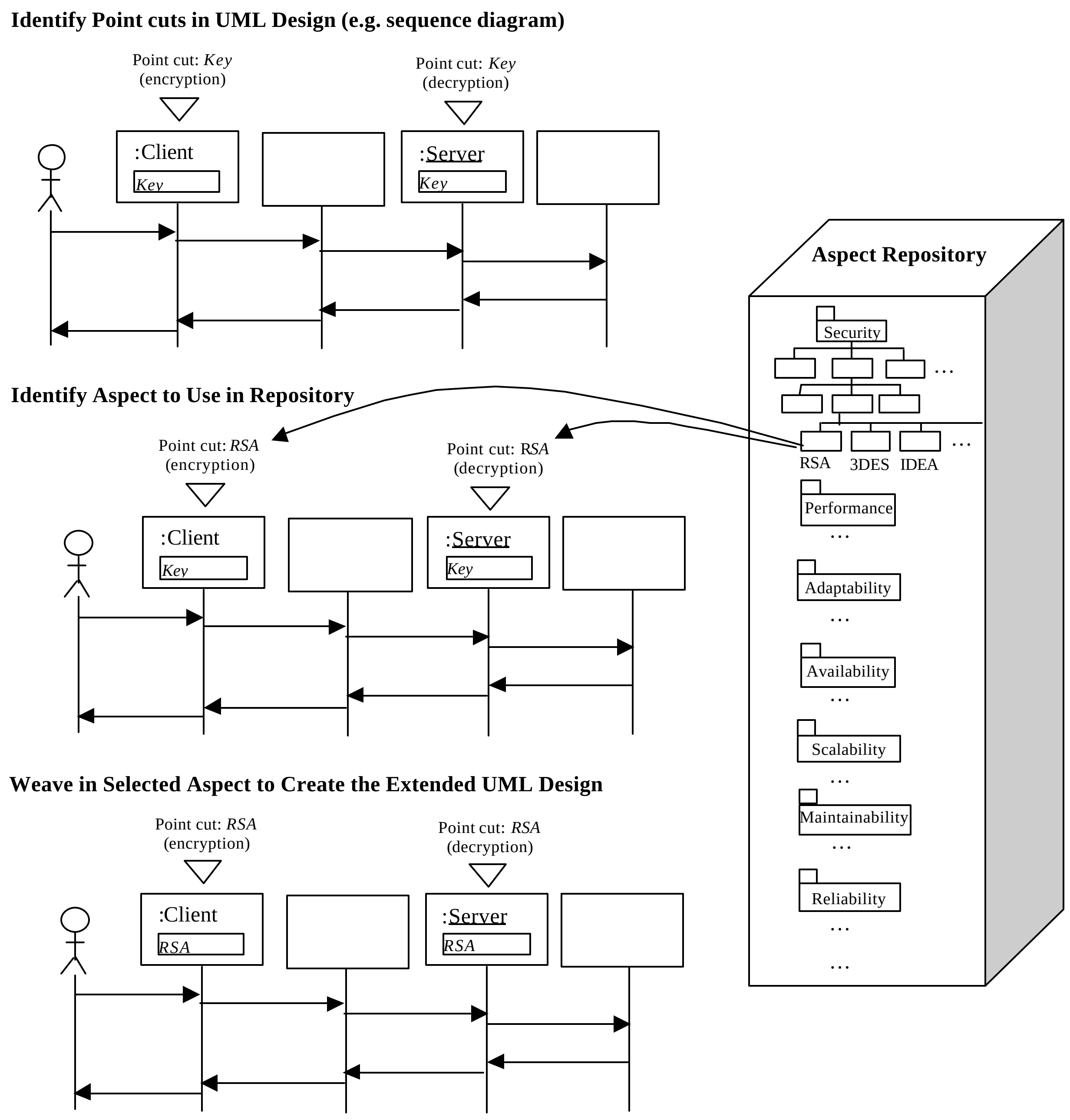

Identify Point cuts in UML Design (e.g. sequence diagram)
Point cut: Key (encryption)
Point cut: Key (decryption)
:Client
Key
:Server
Key
Aspect Repository
Security
RSA
3DES
IDEA
Performance
Adaptability
Availability
Scalability
Maintainability
Reliability
Identify Aspect to Use in Repository
Point cut: RSA (encryption)
Point cut: RSA (decryption)
:Client
Key
:Server
Key
Weave in Selected Aspect to Create the Extended UML Design
Point cut: RSA (encryption)
Point cut: RSA (decryption)
:Client
RSA
:Server
RSA

Figure 4. Creating the Extended UML Design Model

Who performs activity: Formal Methodologists, Specialized Engineers

Description:

Step 3.1 The formal methodologists and specialized engineering (e.g. performance or security engineering specialists) analyze the formal models using existing tool support. For example, if the formal notation Promella is used then the model checker SPIN is used. If the results of the analysis indicate that the non-functional requirements are met, then the process is complete. The designers may need to iterate through this process to consider additional non-functional requirements. However, if the results indicate the non-functional requirements are not met, then the designer needs to revise the extended UML aspect-oriented design and go through the process again.

5. Illustration: Create a Semi-formal Extended UML Aspect-oriented Design Model

This section presents an example illustrating the first step in the FDAF process for a DNS example. The base models for the DNS in UML, the aspect-oriented model for response time performance in the extended UML, and the analysis rules for response time performance are presented. The base models, defined in standard UML, are the input to the FDAF process. In the future, examples are planned to illustrate extending the DNS base model with a security aspect, encrypting and encrypting data to provide data privacy as well as to demonstrate how the framework is used to model the addition of two tangled, conflicting aspects: performance and security.

5.1 Base Models of the DNS

The DNS provides a way to find an address, such as IP address, from a domain name. IP addresses uniquely identify every computer on the Internet. However, remembering 32 bits numeric address is hard. Therefore, the purpose of the DNS is to make it easier for users to access and remember the names of hosts on the Internet. The 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.

The main task of name servers is answering queries using their database they maintain. The data name servers manage in sets are called zones, which include local zones and foreign zones; each zone is the complete database for a particular "pruned" subtree of the domain space. Local zones are loaded from the name server's master files and foreign zones are from other authoritative servers. A name server periodically checks to make sure that its zones are up to date, and if not, obtains a new copy of updated zones from master files stored locally or in another name server.

5.1.1 DNS Use Case Model

As described, besides processing queries from clients, a DNS server needs to refresh its foreign zone periodically from other corresponding authoritative DNS serves. In addition, a DNS server should provide database maintenance functionality for administrators. Functionalities provided by a DNS name server can be described by Figure 5:

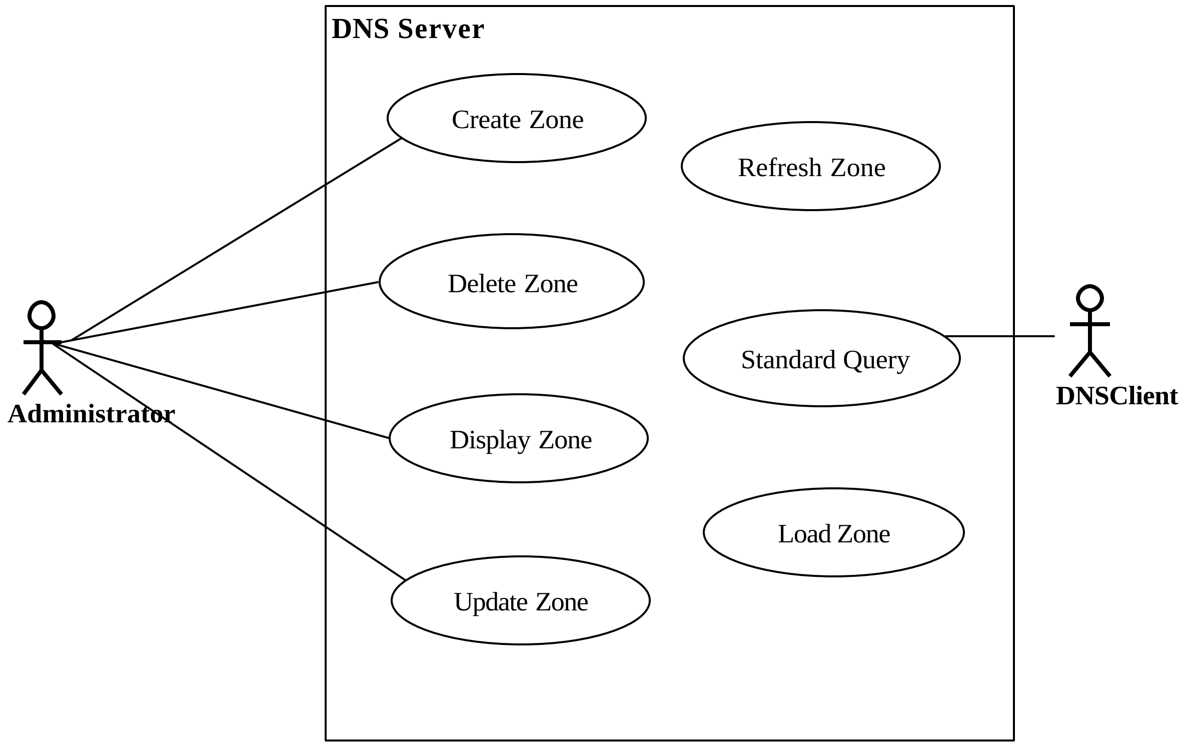


Figure 5 DNS Server Use Case Diagram

5.1.2 DNS Architecture Design Model

Subsystems of the DNS server as well as interfaces they provide are presented in Figure 6. Detailed description of subsystems are described in Table 1.

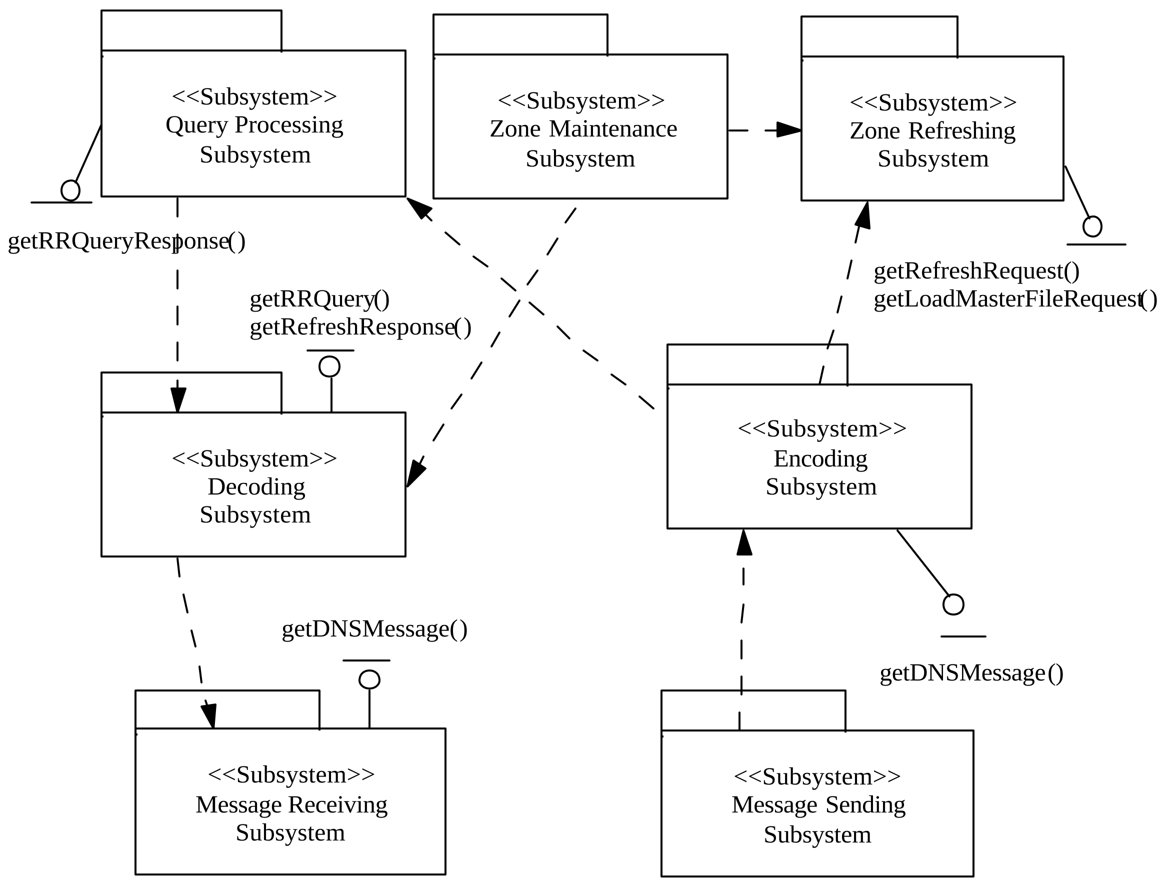


Figure 6 DNS Server Subsystems Diagram

Subsystem Name	**Interfaces Provides**	**Subsystems depends**
Message Receiving Subsystem	*getDNSMessage():* generates a messages in the DNS protocol message format for its received messages;	N. A.
Decoding Subsystem	*getRRQuery()*: generate a client resource record query from a DNS protocol message; *getRefreshResponse():* generate a refresh response sent by other DNS servers from a DNS protocol message;	*Message Receiving Subsystem*: gets a DNS protocol message from this subsystem;
Query Processing	*getRRQueryReponse():* generate answers for a particular client query;	*Decoding Subsystem*: gets client queries from this subsystem;
Zone Maintenance Subsystem	N. A.	*Decoding Subsystem:* gets refresh responses from this subsystem; *Zone Refreshing Subsystem*:

		gets loading master files request from this subsystem;
Zone Refresh Subsystem	*getRefreshRequest():*generates a refresh request for other DNS servers; *getLoadMasterFileRequest()*: generates loading master files requests;	N.A.
Encoding Subsystem	*getDNSMessage()*:generates a encoded DNS protocol message;	*Query Processing Subsystem, Zone Refresh Subsystem:* gets resource records responses and refresh requests for these two subsystems respectively
Message Sending Subsystem	N.A.	*Encoding Subsystem*: gets a encoded DNS message from this subsystem

Table 1. DNS Server Subsystems Description

5.1.3 DNS Class Diagram Models

Figure 7 presents the DNS entity class diagram. Descriptions of classes are available in Table 2.

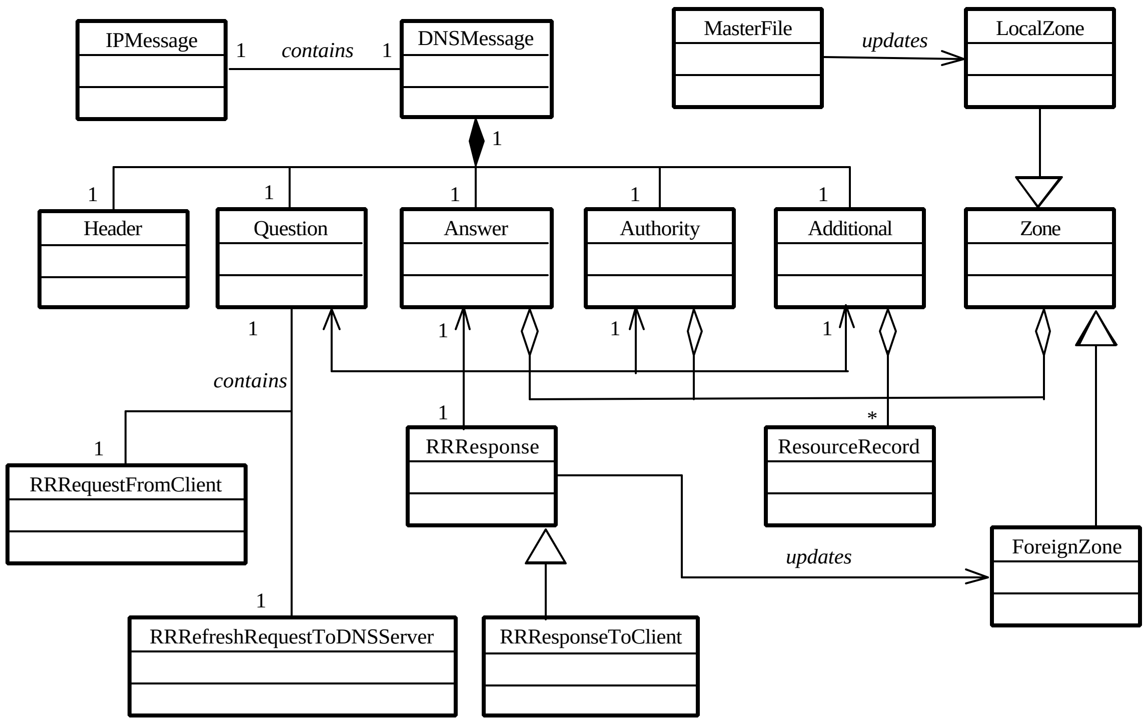


Figure 7. DNS Entity Class Diagram

Class Name	**Description**
Additional	the *Additional* section of the DNS protocol message, which may have zero or multiple resource records
Answer	the *Answer* section of the DNS protocol message, which may have zero or multiple resource records
Authority	the *Authority* section of the DNS protocol message, which may have zero or multiple resource records
DNSMessage	the DNS protocol message
ForeignZone	foreign zone of the DNS server, which needs to be updated by resource records of other DNS servers
Header	the *Header* section of the DNS protocol message
IPMessage	messages that received and sent by the DNS server
LocalZone	foreign zone of the DNS server, which needs to be updated by the DNS server's master file
MasterFile	the DNS server's master file
Question	the *Question* section of the DNS protocol message
ResourceRecord	resource record
RRRefreshRequestToDNSSever	refresh request generated by the DNS server
RRRequestFromClient	resource record query generated by DNS clients
RRResponse	the answer of the DNS server's refresh request
RRResponseToClient	the answer of DNS clients' query
Zone	zone of the DNS server

Table 2. DNS Entity Classes Description

Figure 8 presents another class diagram of the DNS. In this diagram, several control classes have been added in. As the query processing functionality of the DNS is of high interest in this report, the figure only presents control objects involved in this scenario.

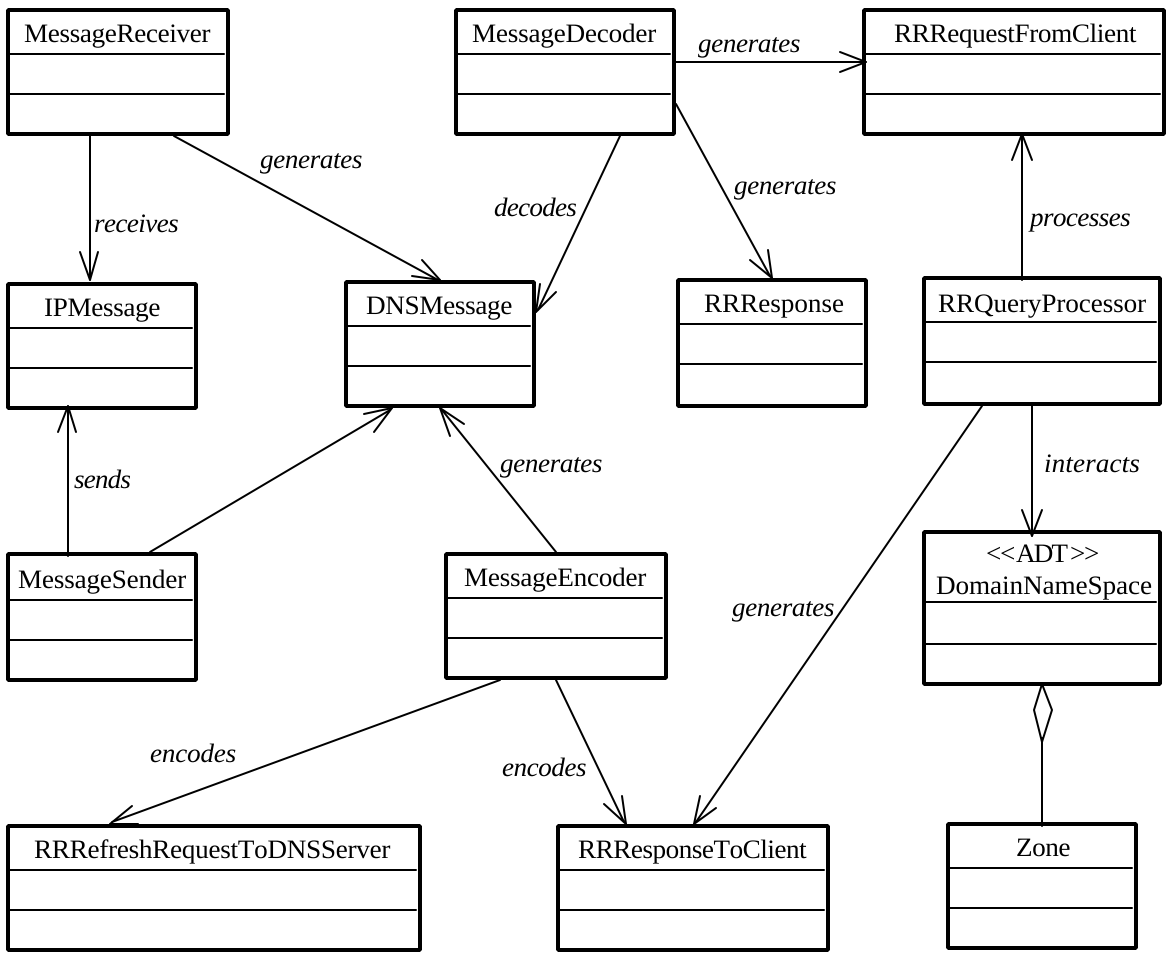


Figure 8. DNS Class Diagram

5.1.4 DNS Query Processing Sequence Diagram

The DNS query processing sequence diagram is presented in Figure 9.

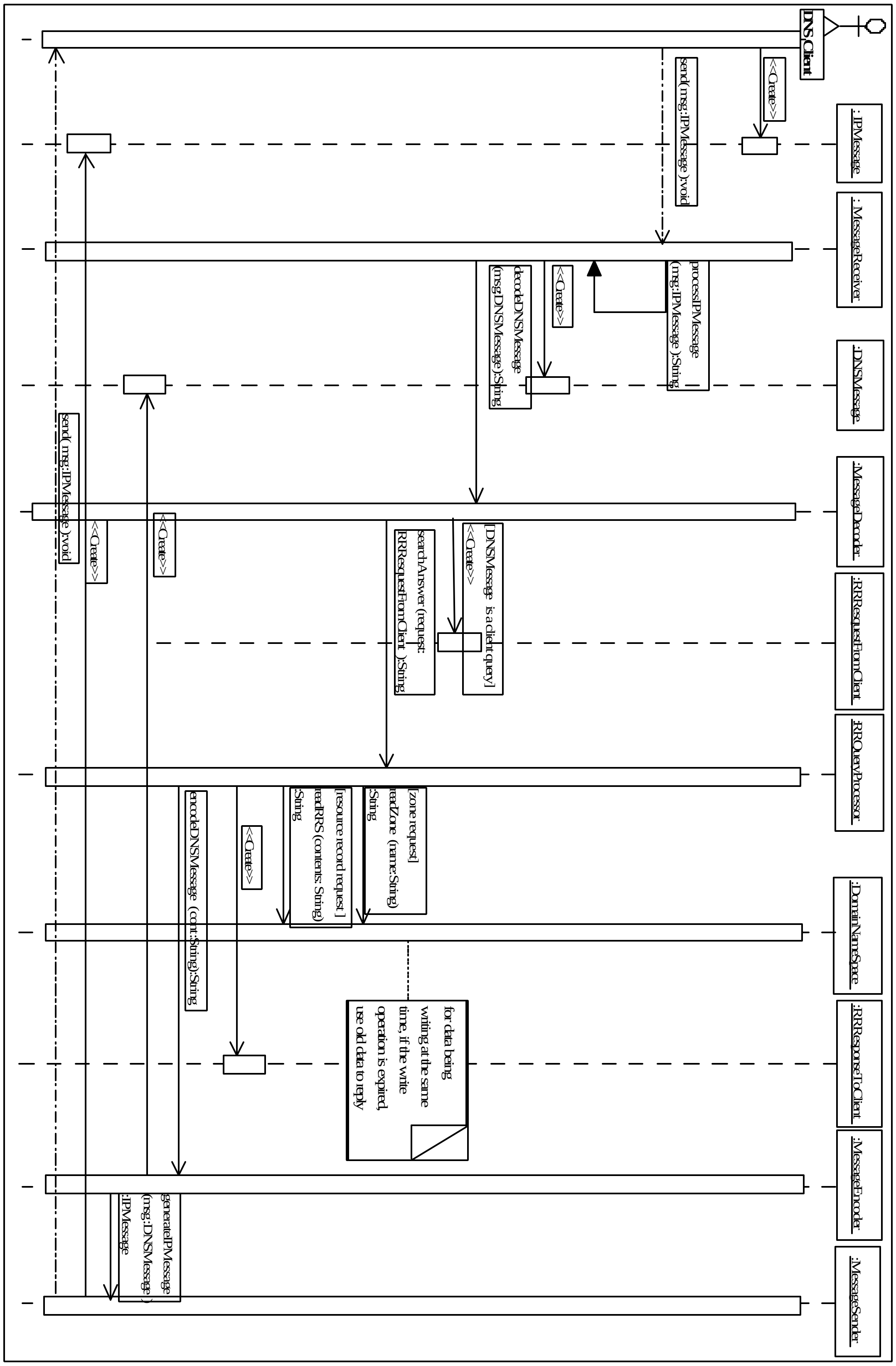

DNS_Client
: IPMessage
: MessageReceiver
:DNSMessage
:MessageDecoder
:RRRequestFromClient
:RRQueryProcessor
:DomainNameSpace
:RRResponseToClient
:MessageEncoder
:MessageSender
<<Create>>
send(msg:IPMessage):void
processIPMessage (msg:IPMessage):String
<<Create>>
decodeDNSMessage (msg:DNSMessage):String
[DNSMessage is a client query] <<Create>>
searchAnswer (request: RRRequestFromClient):String
[zone request] readZone (name:String) :String
[resource record request] readRRS (contents: String) :String
<<Create>>
encodeDNSMessage (cont:String):String
for data being writing at the same time, if the write operation is expired, use old data to reply
generateIPMessage (msg:DNSMessage) :IPMessage
<<Create>>
<<Create>>
send(msg:IPMessage):void

Figure 9. DNS Query Processing Sequence Diagram

5.2 Adding a Performance Aspect

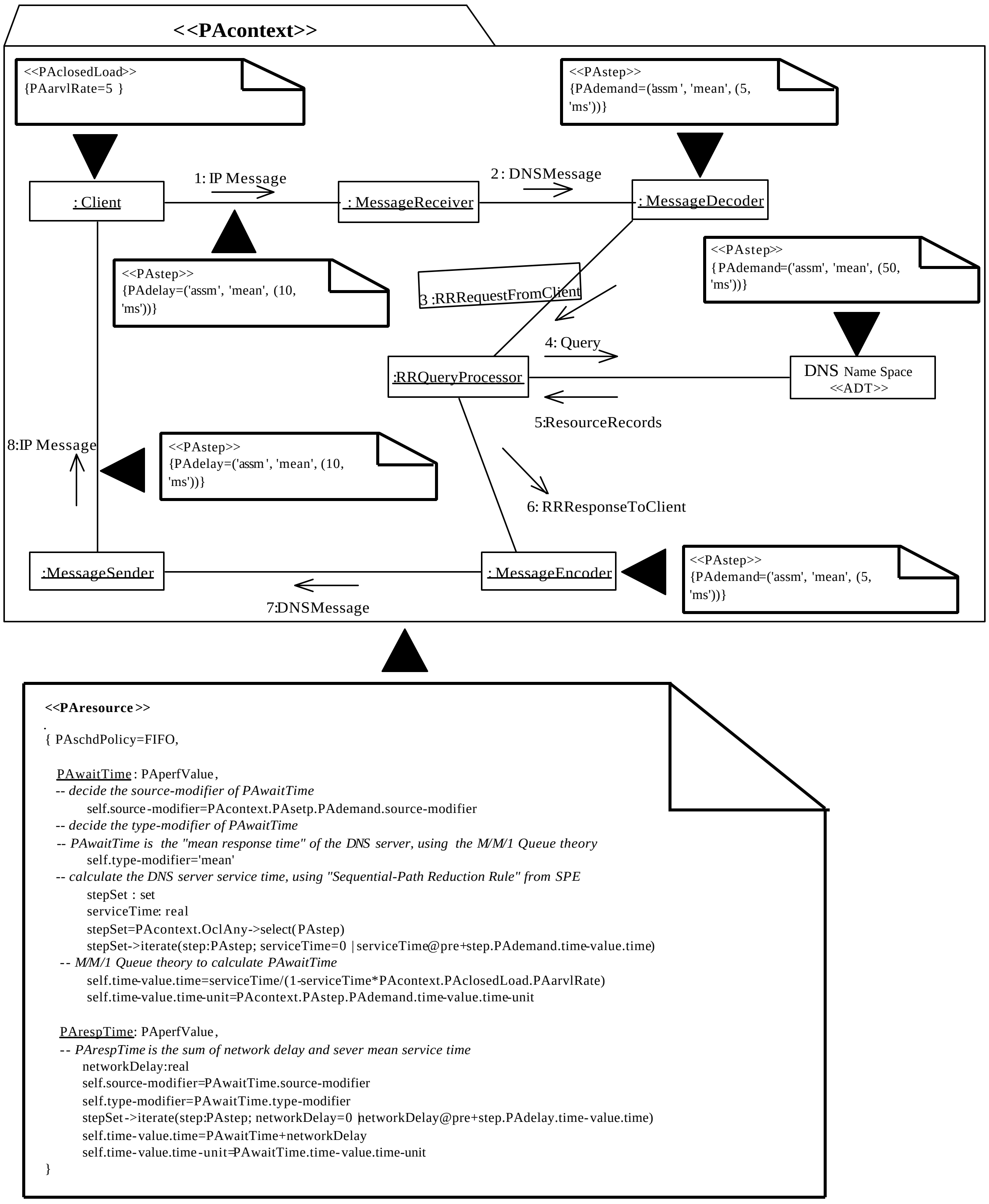
The first example non-functional aspect selected for the DNS is performance aspect. Performance may refer to the response time, system's throughput, as well as efficient utilization of system resources. It is a vital factor in the success of software systems. To meet performance objectives, performance needs to be considered in an early stage of the software development life cycle [23], [37], [48].

The current version of UML, version 1.5, does not support the definition of an aspect. In the FDAF, a new UML extension is presented for this purpose. The UML extension of the FDAF has following two characteristics:

1. Serves as a general-purpose mechanism to model aspects/concerns, which includes performance aspect and other aspects as well;
2. Helps designers to automatically transform the UML semi-formal design models into appropriate formal models and then select the analysis tools in a later stage of the FDAF.

One of the initial steps of performance analysis of software systems is identifying the critical performance scenarios [22], [39], [48]. A scenario is a sequence of actions performed by a group of different objects. Therefore, performance assessment of a key scenario inevitably crosscuts the design model's individual elements. As the main task of a DNS server is to answers queries form clients, the query processing scenario is identified as the key scenario discussed here. Generally, performance assessment includes the evaluations of both resource utilization and response time. As in this example, the response time is the part of the interest.

An example of our UML extension for modeling the performance aspect of the DNS query processing scenario (the collaboration diagram is selected to illustrate the approach) is presented in Figure 10. In our UML extension, a wedge-like, triangular symbol is used to indicate the performance aspect. For each object in an execution path, the performance impact is assigned. The performance value is expressed by using stereotypes, tags defined in [39] and then the UML note notation associates them with the corresponding objects. Outside the package, another note notation is used to denote the performance evaluation techniques that are expressed in the OCL. Performance modeling and analysis techniques of the UML extension are explained in detail below.


<<PAcontext>>
<<PAclosedLoad>> {PAarvlRate=5 }
<<PAstep>> {PAdemand=('assm', 'mean', (5, 'ms'))}
: Client
1: IP Message
: MessageReceiver
2: DNSMessage
: MessageDecoder
<<PAstep>> {PAdelay=('assm', 'mean', (10, 'ms'))}
3 :RRRequestFromClient
<<PAstep>> {PAdemand=('assm', 'mean', (50, 'ms'))}
4: Query
:RRQueryProcessor
DNS Name Space <<ADT>>
5:ResourceRecords
8:IP Message
<<PAstep>> {PAdelay=('assm', 'mean', (10, 'ms'))}
6: RRResponseToClient
:MessageSender
: MessageEncoder
<<PAstep>> {PAdemand=('assm', 'mean', (5, 'ms'))}
7:DNSMessage

```
<<PAresource>>
.
{ PAschdPolicy=FIFO,

  PAwaitTime: PAperfValue,
  -- decide the source-modifier of PAwaitTime
        self.source-modifier=PAcontext.PAsetp.PAdemand.source-modifier
  -- decide the type-modifier of PAwaitTime
  -- PAwaitTime is the "mean response time" of the DNS server, using the M/M/1 Queue theory
        self.type-modifier='mean'
  -- calculate the DNS server service time, using "Sequential-Path Reduction Rule" from SPE
        stepSet : set
        serviceTime: real
        stepSet=PAcontext.OclAny->select(PAstep)
        stepSet->iterate(step:PAstep; serviceTime=0 | serviceTime@pre+step.PAdemand.time-value.time)
  -- M/M/1 Queue theory to calculate PAwaitTime
        self.time-value.time=serviceTime/(1-serviceTime*PAcontext.PAclosedLoad.PAarvlRate)
        self.time-value.time-unit=PAcontext.PAstep.PAdemand.time-value.time-unit

  PArespTime: PAperfValue,
  -- PArespTime is the sum of network delay and sever mean service time
       networkDelay:real
       self.source-modifier=PAwaitTime.source-modifier
       self.type-modifier=PAwaitTime.type-modifier
       stepSet->iterate(step:PAstep; networkDelay=0 |networkDelay@pre+step.PAdelay.time-value.time)
       self.time-value.time=PAwaitTime+networkDelay
       self.time-value.time-unit=PAwaitTime.time-value.time-unit
}
```

Figure 10. DNS Performance Aspect Modeling in UML

5.2.1 Performance Aspect Modeling

The UML extension of the FDAF is based on the principles discussed in [39] and [46]. Software performance engineering (SPE) [46] is a method for constructing software systems to meet performance objectives. SPE methods cover performance data collection, quantitative analysis techniques, prediction strategies, management of uncertainties etc. Data required to evaluate software performance in SPE are: performance goals, workload specifications, software execution structure, execution environment and resource usage. Furthermore, [39] describes a component of the profile that is intended for general performance analysis of UML models. Several performance analysis concepts discussed in [39] are:

- Scenarios--define response paths whose end points are externally visible;
- scenario steps--are elements of scenarios;
- resource demands--the execution time taken on its host device by a step;
- resources--simply servers;
- performance measures--includes resource utilization, waiting times, execution demands and response time, which may be specified in four different versions, namely, “required”, “assumed”, “estimated” and “measured”.

In [39], stereotypes are used to represent performance analysis concepts. To minimize the possibility of confusion and conflict with other UML profiles, all extension element names related to performance analysis are prefixed with the string “PA”. This naming convention has been adopted by the UML extension of the FDAF. However, in our UML extension, the prefix “PA” may also mean “Performance Aspect” related elements. Additionally, [39] defined possible tags to associate with each stereotype.

In our DNS example (see Figure 10), we used following stereotypes that already defined in [39]:

- *<<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 in [39]: *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, which is also used in the DNS example;
- *<<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). In our example, we select the tag *PAdemand* for those steps inside the query processing scenario. For message transmission steps, we select *PAdelay* tag for them;
- *<<PAresource>>*--models a passive resource and has tags: *PAuilization* (utilization), *PAschdPolicy* (scheduling policy), *PAcapacity* (capacity), *PAaxTime* (access time), *PArespTime* (response time), *PAwaitTime* (wait time), and *PAthoughtput* (throughtput). In Figure , this stereotype is used to model the whole performance of the DNS server’s query processing. As response time is of the interest, tags *PArespTime* and *PAwaitTime* are selected. Their values are decided by applying OCL rules (recorded in the note notation outside of the collaboration diagram) to tag values of *PAclosedLoad* and

PAstep specified in the diagram by the designer. Details about those OCL rules are explained in the next section.

In order to make performance analysis meaningful, a new type, *PAperfValue*, is defined in [39] to specify a complex performance value, which includes not only numerical values for performance-related characteristics but also the semantics of those values (e.g., average, maximum, prediction, measurement). The value type is an array in the following format:

"("<source-modifier>", " <type-modifier>", " <time-value>")"

Where:

- *<source-modifier>*::= 'req' | 'assm' | 'pred' | 'msr', is a string that defined the source of the value meaning respectively: *required, assumed, predicted,* and *measured;*
- *<type-modifier>*::= 'mean' | 'sigma' | 'kth-mom', <Integer> | 'max' | 'percentile', <real> |'dist', is a specification of the type of value meaning: *average, variance, k^{th} moment, maximum, percentile range*, or *probability distribution*;
- *<time-value>* is a time value which has two parts: numerical time and time unit.

Several tags used in the DNS example take *PAperfValue* type, such as *PAdemand*. For example, *{PAdemand = {'msr', 'mean', (20,'ms'))}* represents a demand in a scenario step with a measured mean value of 20 milliseconds.

As one important motivation of the FDAF UML extension is to assist designers in analyzing performance, the OCL is selected for this purpose at current stage. In order to conveniently express the *PAperfValue* type in OCL, the three part of its value (source-modifier, type-modifier, time-value) could be seen as its three attributes. The same consideration has been applied to the "time-value", since it includes numerical time and time unit two parts. Therefore, time-value has two attributes: time and time-unit. For example, *PAdemand.time-value.time* refers to the execution time of a particular *PAdemand*.

5.2.2 Performance Aspect Analysis

In our extension, performance analysis rules are expressed by using the OCL and captured in the note notation (see Figure 10). These rules are based on performance analysis techniques from [46] and [21] and focus on the response time for a service request.

Software performance engineering (SPE) [46] is a work that prescribes algorithms for quantitative performance analysis. Algorithms formulated in SPE are for evaluating execution graphs. However, these algorithms are very understandable and can be applied to many types of software models, even a textual description of the system's execution structure.

Here, we present three basic performance analysis algorithms from [46]:

- *Sequential-Path Reduction Rule*: this rule denotes that for a sequential path in the system's execution structure, the time for the computation of the (i+1)st step is the sums of times in sequence in the ith step, which is $t_1^{i+1} = \sum_{j=1}^{n} t_j^i$;

- *Repetition-Path Reduction Rule*: this rule denotes that for a repetition path in the system's execution structure, the time for the computation of the (i+1)st step is the result of multiplying the time of ith step by the loop repetition factor, which is $t_1^{i+1} = nt_1^i$;
- *Conditional-Path Reduction Rule*: this rule denotes that for a conditional path in the system's execution structure, multiplies each ith step by its execution probability, and adds the time for determining which condition holds, which is $t_1^{i+1} = t_0^i + \sum_{j=1}^{n} p_j t_j^i$;

In the DNS example, we need to compute the mean response time (*PArespTime* of stereotype *PAresource*) of the DNS server. As the query arrival rate is already specified in the stereotype *<<PAclosedLoad>>*, according to the M/M/1 Queue theory (only one server is considered in the design) from [21], we need to find out the service rate, which can be easily calculated through the service time (*PAwaitTime* of stereotype *PAresource*).

Steps involved in computing *PAwaitTime* and P*ArespTime* are:

1. Decide *PAwaitTime*

As *PAwaitTime* has three attributes *(source-modifier, type-modifier, and time-value)* discussed in the previous section, we have to decide these three parts one by one.

Decide source-modifier: Although designers can use any of the four types of *source-modifier* values (required, assumed, predicted and measured), for the purpose of consistent analysis, we strongly recommend that designers should use the same kind of source-modifier through one *<<PAcontext>>*. In this case, the value for *PAwaitTime.source-modifier* is simply the *source-modifier* used in the *<<PAcontext>>*, which is OCL expression:

self.source-modifier=PAcontext.PAsetp.PAdemand.source-modifier;

Decide type-modifier: *PAwaitTime* refers to mean response time here. Therefore its *type-modifier* is "mean":

self.type-modifier='mean';

Decide time-value: *Sequential-Path Reduction Rule* of SPE [46] is applied to compute the service time value as the DNS server processes queries sequentially. *<<PAstep>>s* involved here are those who has demand that executes on the hose and the mean service time is the sum of all the *PAdemand.time-value.time* in this *<<PAcontext>>,* which is OCL expressions:

-- define a set

stepSet : set

-- define a real variable

serviceTime: real

-- select PAstep stereotypes that has demand executes on the host

stepSet=PAcontext.OclAny->select(PAstep)

-- calculate serviceTime

stepSet->iterate(step:PAstep; serviceTime=0 | serviceTime@pre+step.PAdemand.time-value.time)

-- M/M/1 Queue theory to calculate PAwaitTime

*self.time-value.time=serviceTime/(1-serviceTime*PAcontext.PAclosedLoad.PAarvlRate)*

-- time-unit is simply the one used by the designer in the PAcontext

self.time-value.time-unit=PAcontext.PAstep.PAdemand.time-value.time-unit

2. Decide *PArespTime*

PArespTime has the same source-modifier and type-modifier with *PAwaitTime* and is the sum of mean service time and message delays on the network**:**

<u>PArespTime:</u> PAperfValue,

-- PArespTime is the sum of network delay and server mean service time

networkDelay:real

self.source-modifier=PAwaitTime.source-modifier

self.type-modifier=PAwaitTime.type-modifier

-- select PAsteps that involved in network delays

stepSet->iterate(step:PAstep; networkDelay=0 |networkDelay@pre+step.PAdelay.time-value.time)

self.time-value.time=PAwaitTime+networkDelay

self.time-value.time-unit=PAwaitTime.time-value.time-unit

Mean Response Time: Sample Calculation

For example, to calculate the server's mean response time for Figure 10, following steps are performed:

1. Calculate the server's service time, which is the sum of execution time of all those performance contribution objects in this scenario (e.g., those design elements associated with the triangle notation in the diagram). In this case, performance contribution objects identified are: MessageDecoder, RRQueryProcess (which searches answers in the DNS Name Space, the time spending on searching could be viewed as the execution time of RRQueryProcessor), MessageEncoder. Hence, the server's service time = 5 + 50 + 5 = 60 milliseconds;
2. Calculate the server's mean response time without network delay by using M/M/1 queue theory. In this example, the requests' arrival rate is 5 per seconds, therefore, the server's mean response time = $(60*0.001)/(1-60*0.001*5) = 86$ milliseconds;
3. Calculate the total response time. Total response time = server's mean response time + network delay = 86 + 10 + 10 = 126 milliseconds.

6. Conclusions and future work

The main contribution of this work is to define a response time performance aspect that incorporates established, quantitative performance analysis techniques [21][46]. This aspect is used in the Formal Design Analysis Framework (FDAF) to create a semi-formal extended UML aspect-oriented design model. The FDAF is intended to support the design and analysis of multiple, non-functional properties using a combination of existing semi-formal and formal methods. The FDAF integrates current research in aspect-oriented design and uses the concepts of the NFR Framework to support the identification, analysis, and negotiation of conflicting non-functional properties. The goals of the framework include assisting the architect to select an appropriate formal method, identify, analyze, and negotiate conflicting properties, define and maintain formal model for specific aspects of the system (e.g., performance, security, adaptability, etc.), and analyze formal models.

In the FDAF, performance is treated as a collection of aspects modeled in the design. Basically, performance is a function of the frequency and nature of inter-component communication, in addition to performance characteristics of those components themselves [9]. Based one the research on [21] and [46], 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 Events, Time between Event}. As a system could have its own special requirements, performance subaspects are not limited to those listed above. According to the need of a particular application, requirement engineers and designers might define their own performance subaspects of interest and add them into the set.

We have used a DNS subsystem, the query processing system, to illustrate the UML extension in this report. In this example, we select the response time performance aspect as the modeling aspect. Response time is defined as the interval between a user's request and the system response. Stereotypes defined in the UML profile for performance modeling [39] are used here. The calculation of the response time performance aspect using performance analysis rules are expressed by using the OCL and is done manually at current stage. Our approach provides a way for designers to model and analysis interested performance aspect in the design phase, which enables one to evaluate different design alternatives according to some specific performance goals and select the most suitable one for the system to meet its performance requirements.

There are several interesting directions for the future work. One direction includes investigating the real-time extension of UML. This extension has been developed to address specific problems in modeling real-time systems such as concurrency and synchronization and may be a more suitable notation for our example system, the DNS.

A second direction is to investigate the automatic translation of the extended UML DNS design model into the notation RAPIDE and Armani [34]. RAPIDE is an event-based concurrent object-oriented language designed for simulation and behavioral analysis of architectures of distributed systems at an early stage, which also provides timing model to allow designers to describe and analyze time sensitive prototypes. There are several analyzing tools supported by Rapide to simulate a software architecture. Armani is a language for capturing software architecture design expertise and specifying software architecture designs. Armani provides core language constructs to support design analysis and also allows designers to build their own analysis methods. It is used in the ACME tool set [17]. The ACME toolset supports the quantitative performance analysis of an architecture. Depending on the estimated data (such as request arrival rate, service time) provided by the designer, Armani’s analysis tool can automatically evaluate the design’s performance results, such as server mean response time, overloaded component etc. through performance measure techniques presented in [21] and [46]. The advantage of performing such translation work includes that the usage of existing tools supported by Rapide and Armani can help designers to analyze their formal model for a specific system aspect, and provide valuable analysis results for them to evaluate and improve their design before the system is implemented.

A third direction is to investigate the modeling and analysis of additional aspects, such as security, and the interactions among these aspects. The non-functional requirements (NFR) framework [8] is going to be used to systematically analyze the synergistic and conflicting relationships among the aspects. For example, security and performance represent conflicting properties. In general, the more secure a system is, the slower the performance is expected to be, unless alternative solutions such as hardware implementations of encryption/decryption algorithms are considered that are likely to increase the cost of the system.

7. References

[1] Allen, R.J., “A formal approaches to software architecture”, PhD Thesis, Carnegie Mellon Univ., CMU Techinical Report CMU-CS-97-144, May 1997.

[2] Basch, M. and Sanchez, A., “Incorporating aspects into the UML”, Workshop on Aspect-Oriented Modeling with UML, March 2003.

[3] 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.

[4] Bernardo, M., Ciancarini, P., and Donatiello, L., "Detecting architectural mismatches in process algebraic descriptions of software systems", Proc. of Working IEEE/IFIP Conf. on Software Architecture, 2001, pp. 77 - 86.

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

[6] Buchs, D., Guelfi, N., “A formal specification framework for object-oriented distributed systems”, IEEE Transactions on Software Engineering, Vol. 26, Issue 7, July 2000, pp. 635-652.

[7] Chung, L., Cooper, K., and Yi, A., "Developing Adaptable Software Architectures for Real-Time Systems", Proc. of the Int. Conf. on Adapatable Software Architecture, 2002, pp. 43-48..

[8] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing, 2000.

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

[10] Cooper, K., Dai, L., Deng, Y., and Dong, J., "Defining a Formal Design Analysis Framework", Proceedings of SERP 2003, Las Vegas, Nevada, to appear.

[11] Dong, J., “Representing the Applications and Compositions of Design Pattern in UML”, Proceedings of the ACM Symposium on Applied Computing (SAC), Melbourne, Florida, USA, March 2003.

[12] Eastlake, D., “DNS Security Operational Considerations”, RFC2541, March 1999.

[13] Elrad, T., Filman, R.E., Bader, A., “Aspect-oriented programming: Introduction”, Communications of the ACM, October 2001, Vol. 44, Issue 10, pp. 29 - 32.

[14] France, R., Georg, G. and Ray, I., “Supporting multi-dimensional separation of design concerns”, Workshop on Aspect-Oriented Modeling with UML (AOSD’2003), March 2003.

[15] Franch, X., ‘Systematic Formulation of Non-Functional Characteristics of Software”,1998 International Conference on Requirements Engineering (ICRE '98) ,April 06 - 10, 1998

[16] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns - Elements of Reusable Object-Oriented Software,* Addison-Wesley, USA, 1995.

[17] Garlan, D., Monroe, R., Wile, D., “ACME: an architecture description interchange language”, Proceedings of CASCON 97, November 1997, pp. 169-183.

[18] He, X. and Deng, Y., "A Framework for Developing and Analyzing Software Architecture Specifications in SAM", Computer Journal, Vol. 45, No. 1, 2002, pp. 111-128.

[19] Ho, W., Jézéquel, J., Pennaneac'h, F., Plouzeau, N., “A toolkit for weaving aspect oriented UML designs”, Proc. of the 1st Int. Conf. on Aspect-oriented software development, April 2002, pp. 99 – 105.

[20] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, USA, 1999.

[21] Jain, R., The Art of Computer Systems Performance Analysis, Wiley, 1991.

[22] 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.

[23] 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.

[24] Klein, M., Kazman, R., “Attribute-Based Architectural Styles”, CMU/SEI-99-TR-22, Software Engineering Institute, Carnegie Mellon University, 1999.

[25] Lindemann, C., Thümmler, A., Klemm, A., Lohmann, M., and Waldhorst, O.,” Performance analysis of time-enhanced UML diagrams based on stochastic processes”, Proceedings of the Third International Workshop on Software and Performance (WOSP’2002), July 2002, pp. 25-34.

[26] Luckham, D., Kenney, J., Augustin, L., Vera, J., Bryan, D., and Mann, W., "Specification and analysis of system architecture using RAPIDE", IEEE Trans. on Software Engineering, Vol. 21, No. 4, April 1995, pp. 336-355.

[27] Magee, J., Dulay, N., Eisenbach, S., Kramer, J., “Specifying Distributed Software Architectures”, Proc. Fifth European Software Eng. Conf., July 1995, pp. 137 - 154.

[28] Magee, J., Kramer, J., “Dynamic Software in Software Architectures”, Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering, Oct. 1996, pp. 3 - 14.

[29] McUmber, W. E., Cheng, B.H.C., “A general framework for formalizing UML with formal languages”, Proc. of the 23rd Int. Conference on Software Engineering, July 2001, pp. 433 - 442

[30] Medvidovic, N. and Taylor, R.N., "A classification and comparison framework for software architecture description languages", IEEE Trans. on Software Engineering, Jan. 2000, Vol. 26, Issue 1, pp. 70-93.

[31] Medvidovic, N., Rosenblum, D.S., Redmiles, D., F, Robbins, J.E., “Modeling software architectures in the Unified Modeling Language**”,** ACM Transactions on Software Engineering and Methodology (TOSEM), Jan 2002, Vol. 11, Issue 1, pp. 2-57.

[32] Merseguer, J., Campos, J., and Mena, E., “Performance analysis of internet based software retrieval systems using Petri Nets”, Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems**,** July 2001.

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

[34] 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.

[35] Netinant, P., C. A. Constantinides, T. Elrad, M. E. Fayad, "Aspect-Oriented Frameworks: the Design of Adaptable Operating Systems". Poster OOPSLA'2000, October 2000.

[36] Netinant, P., Constantinides, C.A., Elrad, T., Fayad, M.E., 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.

[37] Nixon, B.A., "Management of performance requirements for information systems", IEEE Transactions on Software Engineering, Vol.26, Issue 12, Dec 2000, pp.1122 -1146.

[38] Noda, N., and Kishi, T., "On aspect-oriented design-an approach to designing quality attributes", Proc. 6th Asia Pacific Software Engineering Conference, December 1999, pp. 230-237.

[39] Object Management Group, Response to the OMG RFP for Schedulability, Performance, and Time, OMG Documents ad/2001-06-14, June 2001.

[40] Object Management Group, The Unified Modeling Language (UML), version 1.5, 03-03-01, 2001.

[41] Pinto, M., Fuentes, L., Fayad, M.E., Troya, J.M., "Separation of coordination in a dynamic aspect oriented framework", Proceedings of the 1st international conference on Aspect-oriented software development, April 2002, pp. 134 – 140.

[42] Salazar-Zárate, G., and Botella, P., "Use of UML for modeling non-functional aspects", Proceedings of the 13th International Conference on Software & Systems Engineering and their Applications (ICSSEA'2000), 2000.

[43] Shaw, M., "The coming-of-age of software architecture research", Proc. Int. Conference on Software Engineering, 2001, pp. 656-664.

[44] Simon, H.A., *The Sciences of the Artifical*, The MIT Press, Cambridge, Massachusetts, 1981.

[45] Smarkursky, D.L., Ammar, R.A. and Sholl, H.A., "A framework for designing performance-oriented distributed systems", Proc. 6th IEEE Symposium on Computers and Communications, 2001, pp. 92 - 98.

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

[47] Warmer J. B., and Kleppe, A. G. The Object Constraint Language: Precise Modeling with UML. Addison-Wesley, 1998.

[48] Williams, G. L., and Smith, C. U., "Performance evaluation of software architecture: PASA^{SM}: a method for the performance assessment of software architectures", Proceedings of the Third international workshop on Software and performance**,** July 2002.

Appendix A. Performance Aspect

As described in [21], a performance study needs a set of performance criteria or metrics. One approach to prepare this set is 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 the events can be measured.

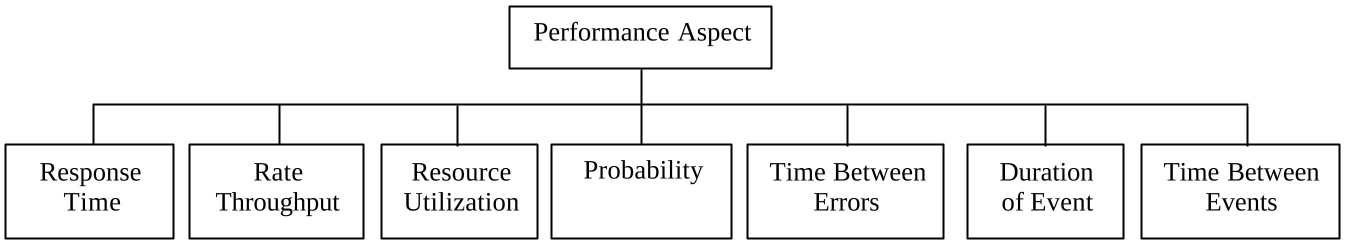


Figure A1. Static View of Performance Aspect (adapted from [21])

Appendix B. Security Aspect

Security is a complex aspect that provides design solutions for access, authentication, accounting, privacy, and integrity. The security aspect is modeled as a set of packages (refer to Figure A1). In this Figure three packages are defined at the most abstract level to contain designs for standard reference models, encryption/decryption algorithms, and protocols. At the lowest level of abstraction, a package's subdiagram is defined with a class diagram, sequence diagram, or collaboration diagram. For example, a sequence diagram for the NIST BSR INCITS draft standard for Role Based Access Control is presented in Figure A2. The security aspect is being refined to include additional designs.

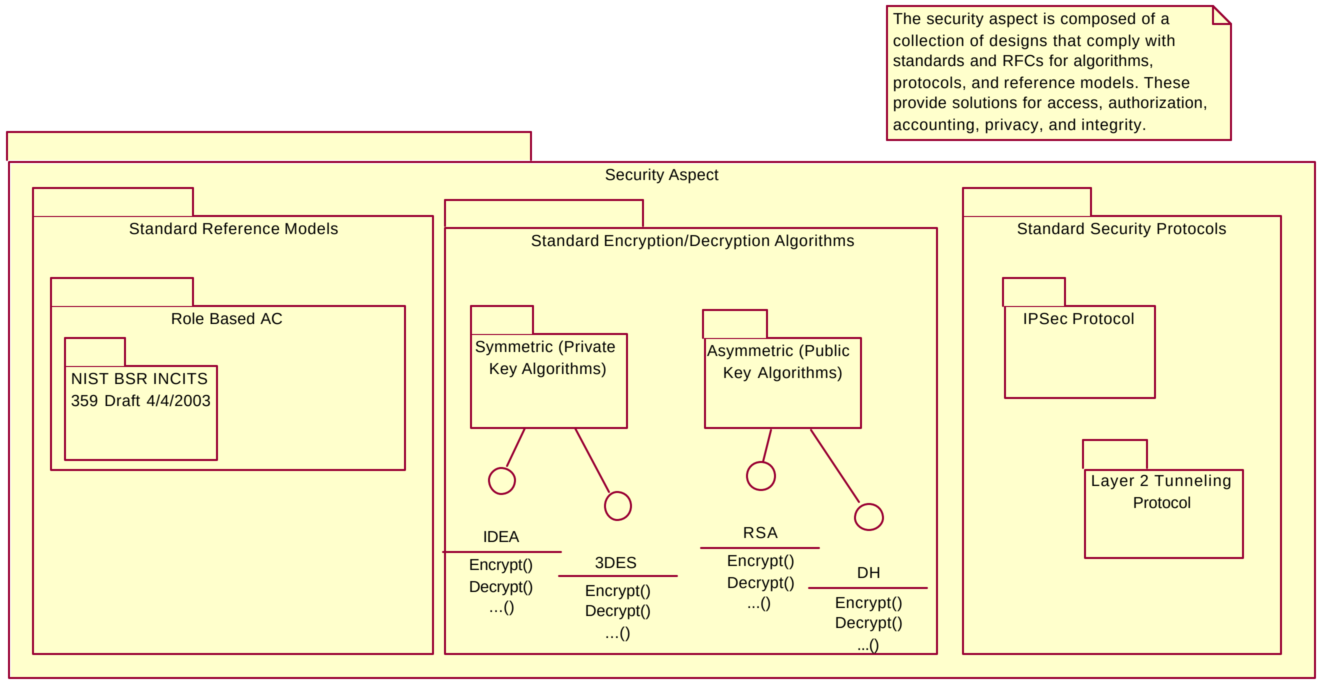

The security aspect is composed of a collection of designs that comply with standards and RFCs for algorithms, protocols, and reference models. These provide solutions for access, authorization, accounting, privacy, and integrity.
Security Aspect
Standard Reference Models
Role Based AC
NIST BSR INCITS 359 Draft 4/4/2003
Standard Encryption/Decryption Algorithms
Symmetric (Private Key Algorithms)
Asymmetric (Public Key Algorithms)
IDEA
Encrypt()
Decrypt()
...()
3DES
Encrypt()
Decrypt()
...()
RSA
Encrypt()
Decrypt()
...()
DH
Encrypt()
Decrypt()
...()
Standard Security Protocols
IPSec Protocol
Layer 2 Tunneling Protocol

Figure B1 . Security Aspect