In the name of God

Ranking Different Software Architectures Based on QoS Using Analytical Models

---

CS672 - Final Project Report

Naeem Esfahani

November 2009

Table of Contents

1 Introduction

Considering non-functional requirements (i.e., Quality of Service) is missing from many software construction paradigms. Many of these paradigms focus on functional requirements and neglect the QoS at design time. In this report we provide an approach to address this problem at design time. In the introduction, first we describe a case study in context of which the problem is described, and then we provide the architecture model for the case study.

1.1 Case Study

Instead of going directly to airlines many people use travel agents to arrange their flights. This led to the introduction of Online Ticketing Systems (OTS) [1,2]. These systems integrate different agents and companies and provide a single interface for the user. User can provide the request to OTS and it will find the matching offers. Figure 1, Shows a high level view of such a systems.

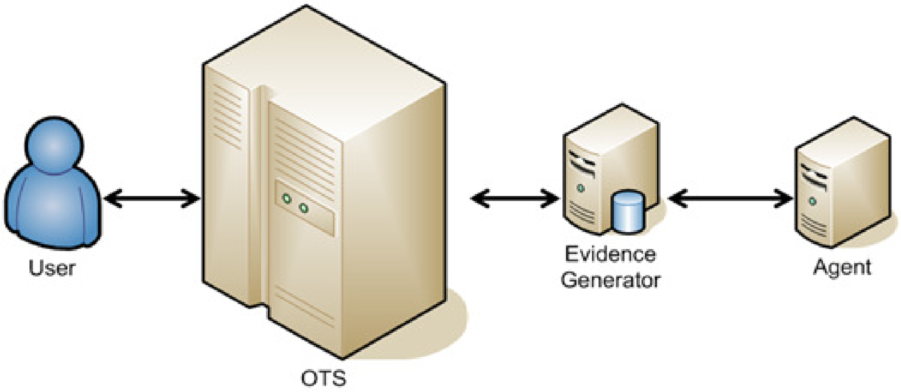


Figure 1. High level view of Online Ticketing Systems

OTS systems usually have a list of known agents. When they cannot answer the request with the known agents, they try to discover new agents. Moreover, to keep track of Service Level Agreements (SLAs) between the system and agents there may be a third party involved in the transaction between the system and agent to record the evidences.

1.2 Software Architecture

Software architecture is usually modeled using an Architecture Description Language (ADL) [3]. Figure 2a shows an implementing architecture for the high level view presented in Figure 1. In this diagram there are four constructs: Component, Connector, Group, and Interface. Components are independent units of activities, while Connectors (not shown in Figure 2) are domain independent facilities for their communication. Groups show the allocation of Components and Connectors into different nodes in the system and Interfaces model how they can interact to each other.

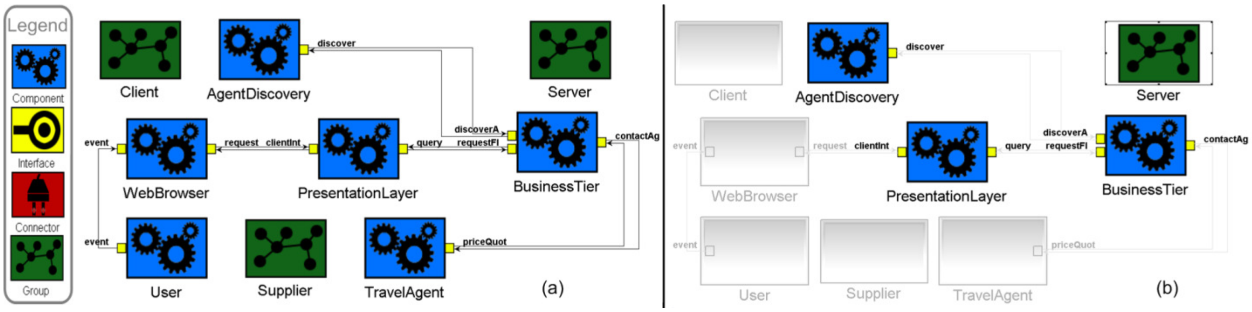


Figure 2. a) Core architectural model for OTS, and b) Server group is selected.

The architectural model depicted in Figure 2, is generated in XTeam [4] which is a simulation framework in the level of software architecture. Figure 2b shows how groups relate different components together.

1.3 Feature Model

One way for decomposing a large system is through features. A feature may correspond to a business use case, resources allocation algorithm, authentication protocol, or any other capability of the system. Also, features keep us independent of the implementation paradigm or how features are realized. Figure 3a shows the feature model for OTS. Figure 3b shows a feature modeling language implemented as a Domain Specific Modeling Language in Generic Modeling Environment (GME) [5].

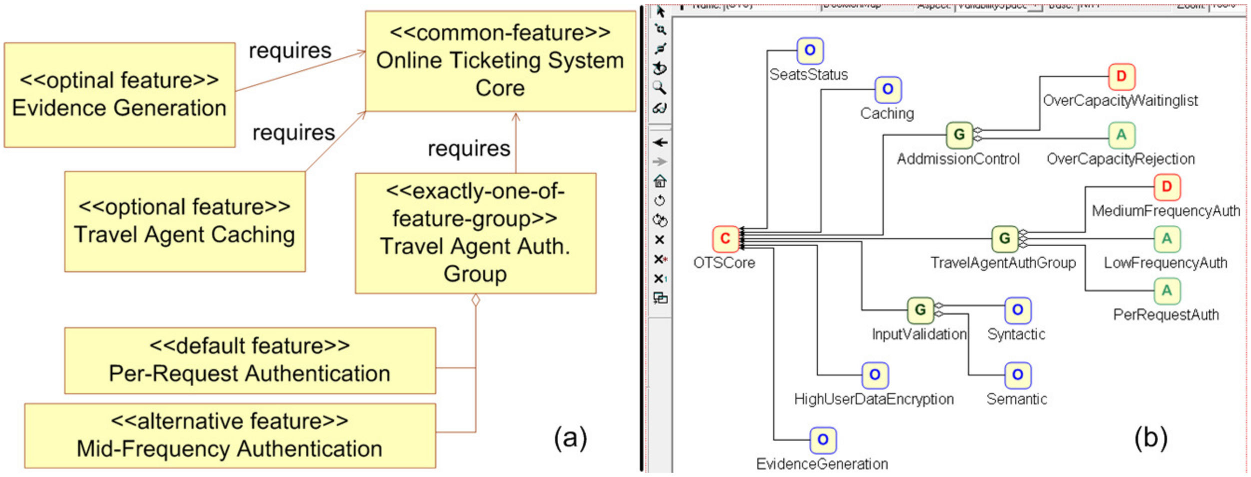


Figure 3. Feature Models. a) abstract feature model, and b) feature model implemented as DSML in GME

Each feature is related to an architectural configuration. All the features are dependent on the Core feature model (annotated with C and expanded in Figure 2). The Arrow from one feature to another one means that the former is dependent on the later. The dependent feature adds functionality to the depended one. Figure 4 shows how Caching feature's architecture is weaved into the Core architecture. When a dependent feature is selected it requires that the depended feature to be selected as well. Using aspect oriented programming [6] it weaves into the depended features architecture and builds new software architecture.

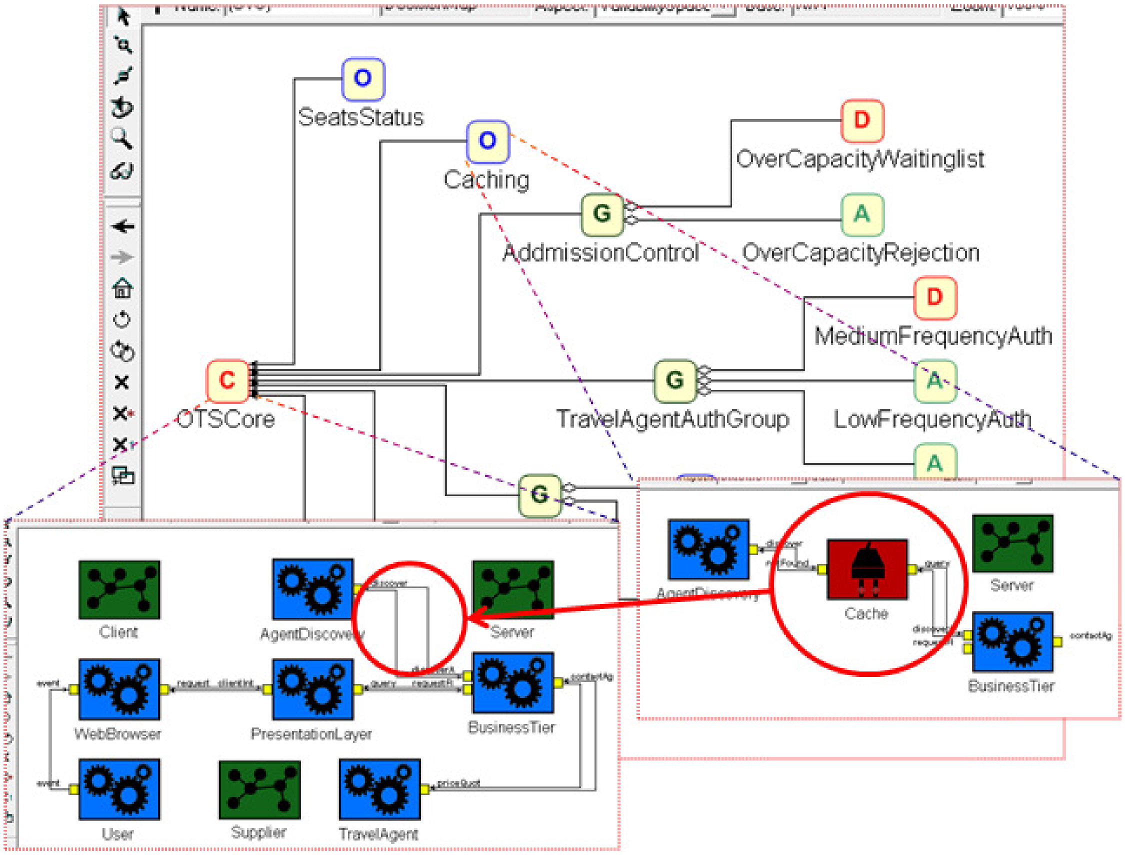


Figure 4. Caching feature's architecture is weaved into the Core architecture.

This approach allows us to automatically build different architectural combinations in design time by selecting different feature combinations. For example, when we select Core, Caching, Evidence Generation, and Per Request Authentications the architecture in Figure 5 will be generated. The generated architecture can be used for simulation in XTeam framework.

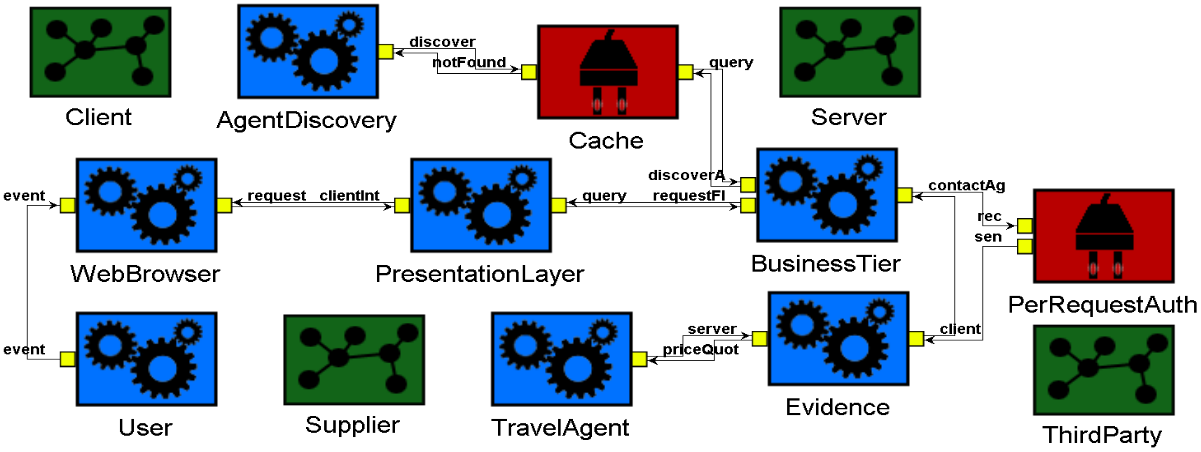


Figure 5. Architectural combination as a result of selecting Core, Caching, Evidence Generation and Per Requeset Authentication features

2 Problem definition

There is a motto that some software engineers were following: “Make It Work, Make It Right, and Then Make It Fast”. This motto is not applicable for all kinds of software and if it is followed everywhere the generated system will be with a very low quality and completely wrong. They are some major decisions that made in earlier phases and cannot be undone easily. These decisions affect the non-functional requirements of the system.

In many cases the designers know how different parts of the system work independently. The question is how good they will work when they are integrated. It would be very effective if the designer can build different configurations for the architecture at design time and using analytical models get an overview about the system. This knowledge can help the administrator of the system to revive it when it fails to provide a QoS by knowing the problematic features priori. For example, if at the pick usage the system failed to provide the result in 6 seconds, which means that the user is going to discard any result, the administrator can lower the security level to revive the system temporarily. This kind of knowledge is not documented in any place and is very human dependent.

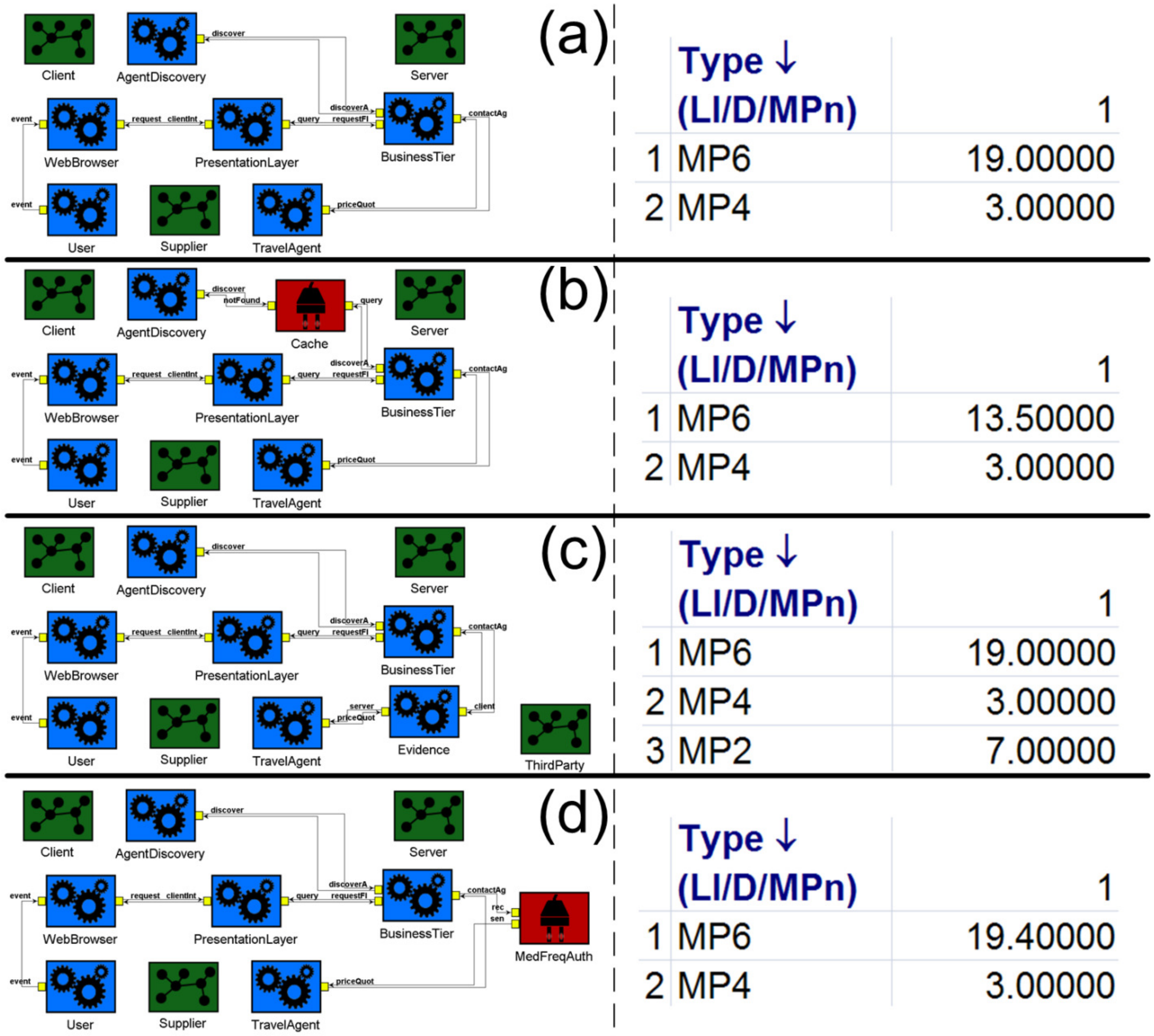
This project is a part of a larger project that tries to address mentioned problem [7]. However, in this project we tried to focus on one QoS requirement in OTS: Response Time. We tried to compare different architectural configurations based on the calculated response time by a queuing network in design time. Different architectural configurations are built by combining different features (recall Section 1.3). The response time is measured as a round trip time from the WebBrowser’s output interface (*request* in Figure 5).

3 Methodology Used

By selecting different legal feature combinations we build different software architectures. These software architectures are later transformed into queuing networks.

3.1 Building Queuing Networks

The input parameters for the Queuing Networks (i.e., Service demands) were harvested from the simulations carried out in the context of different project in CS700 course using XTeam framework. XTeam is a simulation framework in which each group is mapped to a hardware host. Therefore the number of actual resources in the Queuing Network is the number of groups in the architectural model decreased by one due to the fact that the *Client* is just used for generating events. Since the existence of event generator is modeled by number of requests in the Closed Queuing Network the *Client* resource should be dropped from the generated Queuing Network. Figure 6 and Figure 7 summarize the generated Queuing Networks for different architectures. In the left we can see the generated architecture based on the selected features (based on Figure 3a) and in the right we can see the generated Closed Queuing Network.



(a)

Type ↓ (LI/D/MPn)		1
1	MP6	19.00000
2	MP4	3.00000

(b)

Type ↓ (LI/D/MPn)		1
1	MP6	13.50000
2	MP4	3.00000

(c)

Type ↓ (LI/D/MPn)		1
1	MP6	19.00000
2	MP4	3.00000
3	MP2	7.00000

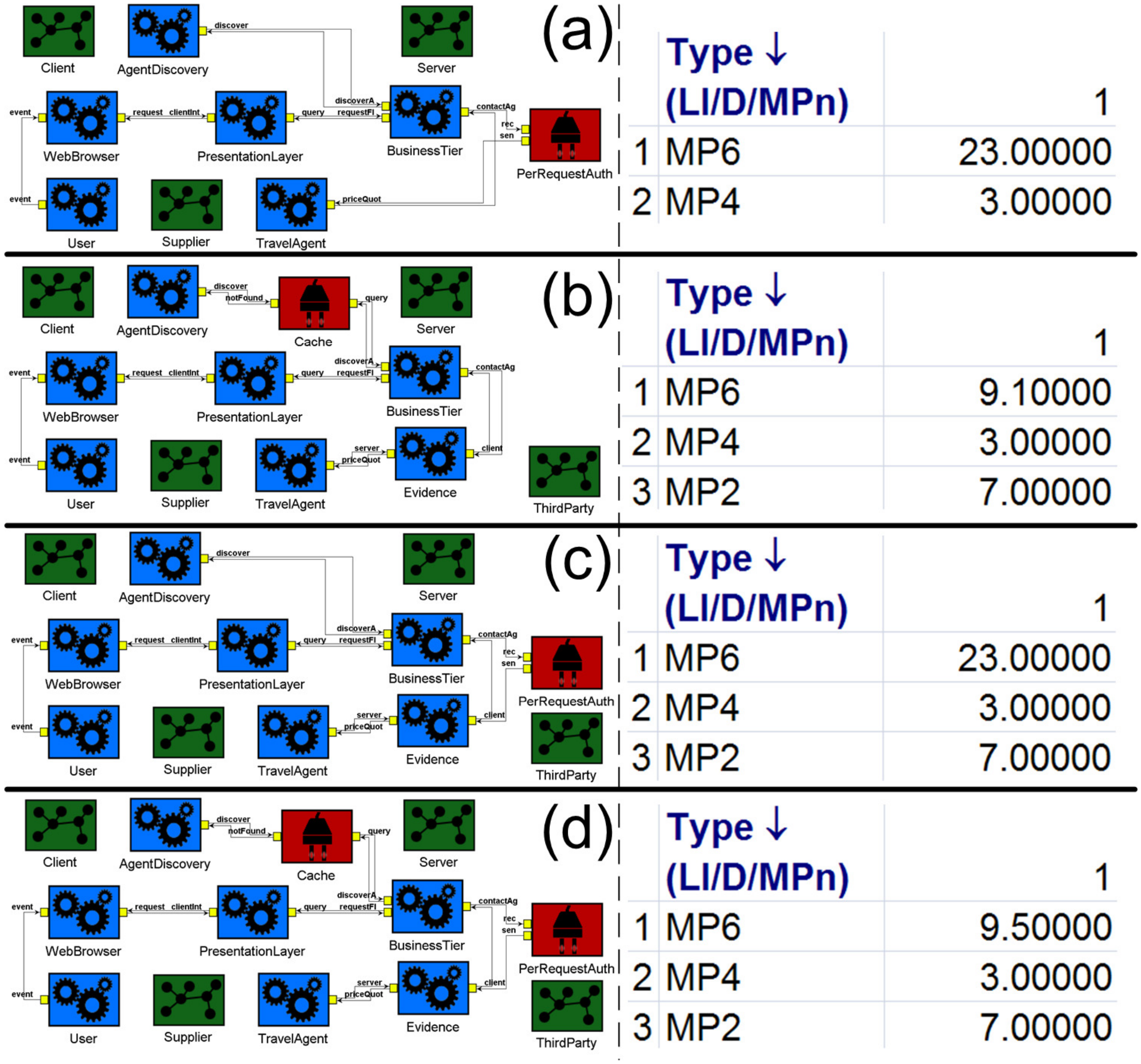
(d)

Type ↓ (LI/D/MPn)		1
1	MP6	19.40000
2	MP4	3.00000

Figure 6. Mapping architecuters to Queuing Networks. Left: architecture generated based on feature selection. Rugh: Queuing Network generated based on the architecture. a) ***Core Architeture*****, b)** ***Core Architecture*** **with** ***Caching*** **enabled, c)** ***Core Architecture*** **with** ***Evidence Generation*** **enabled, d)** ***Core Architecture*** **with** ***Medium Frequency Authentication*** **enabled.**

As depicted in Figure 6 and Figure 7 the resources are considered to be only Multi-CPUs. This is due to the fact that the service demands are provided from the simulation and capture the total amount of resource usage on a group. Since number of threads in a group is similar to having several replicas of a single hardware resource, we can amortize all the resource usage as a single number and model it as CPU usage. Finally, the service demands are the average time spent in each group obtained from the simulations.

Another way for modeling these architectures is to use “Two-Level Iterative Queuing Modeling of Software Contention”; however, since we already had the service demands from the simulations we didn’t use that method.



(a)

Type ↓ (LI/D/MPn)	1
1 MP6	23.00000
2 MP4	3.00000

(b)

Type ↓ (LI/D/MPn)	1
1 MP6	9.10000
2 MP4	3.00000
3 MP2	7.00000

(c)

Type ↓ (LI/D/MPn)	1
1 MP6	23.00000
2 MP4	3.00000
3 MP2	7.00000

(d)

Type ↓ (LI/D/MPn)	1
1 MP6	9.50000
2 MP4	3.00000
3 MP2	7.00000

Figure 7. Mapping architecuters to Queuing Networks. Left: architecture generated based on feature selection. Rugh: Queuing Network generated based on the architecture. a) ***Core Architeture*** **with** ***Per Request Authentication*** **enabled, b)** ***Core Architecture*** **with** ***Caching*** **and** ***Evidence Generation*** **enabled, c)** ***Core Architecture*** **with** ***Evidence Generation*** **and** ***Per Request Authication*** **enabled, d)** ***Core Architecture*** **with** ***Caching, Evidence Generation*** **and** ***Per Request Authication*** **enabled.**

Before presenting the results, it is worth elaborating on one issue; if one looks closely into Figure 6 and Figure 7, he will realize that not all the feature combinations of Figure 3a are modeled there. This is due to the fact that not all the feature combinations make sense, are legal or important. This result obtained from the other part of research [7]. Therefore in this project we didn't investigate all the combination and the results shown in the following section are not going to contain the omitted feature combinations.

3.2 Obtained Results

Table 1 shows the result of executing Queuing Networks of Figure 6 and Figure 7. The left part of the table shows the feature combinations (enabled features) for which the result is shown. In the right side of the table the actual results, which are Response Time and Throughput, are shown. We used Excel worksheets coming with the book [8] to carry out the experiments. Since the models contained multiprocessing elements the original model was changed by the worksheet to generate temporary approximate models. These models were used for solving.

Table 1. Response time and througput obtained from models corresponding to different feature combinations

Features				Response Time (sec)	Throughput
Caching	Evidence	MedFreqAuth	PerReqAuth		
Disabled	Disabled	Disabled	Disabled	25.27848	0.20
Enabled	Disabled	Disabled	Disabled	18.72444	0.27
Disabled	**Enabled**	Disabled	Disabled	33.50943	0.15
Disabled	Disabled	**Enabled**	Disabled	25.75680	0.19
Disabled	Disabled	Disabled	**Enabled**	30.06996	0.17
Enabled	**Enabled**	Disabled	Disabled	24.41048	0.20
Disabled	**Enabled**	Disabled	**Enabled**	37.74949	0.13
Enabled	**Enabled**	Disabled	**Enabled**	24.71799	0.20

4 Analysis of the results

By looking at Table 1 we can rank different feature combinations in terms of selected QoS dimension (i.e., response time). These data can be used to infer the effect of each feature in the response time. As mentioned before this process is the focus of other part of research.

Simply by looking at the results we can say Caching reduces the response time and other features increase it as it was expected. Moreover, we can see the relative impact of each feature on the response time as well. However, for getting more knowledge out of result we need more deliberate analysis to consider all the interrelationships between features (which are called factors in experiment design terminology).

There are several ways for capturing the effect single feature on the response time. One can use learning methods to do that e.g., simple regression techniques can be applied. We can even use classic methods for analyzing the result e.g., ANOVA can be used to see if each feature selection (factor) is really makes the difference. However, the detailed analysis of the results is out of the scope of this project. Here the emphasis of the project is on the whole approach: generating analytical models from the software architecture.

5 Concluding Remarks

In this report we show how feature oriented approaches can be used to build software architecture for building software systems. We further showed a transformation from the software architecture to Queuing Networks. This was done using the results obtained from the simulation of the architecture in XTeam framework (i.e., getting the service demands and number of concurrent requests inside the network from the simulation results).

These models are very helpful when the system is in design phase and an actual system does not exist to be monitored. Analytical models can be used to approximate the way system behaves. However, building analytical models is a little bit hard and needs knowledge of mathematics. Unfortunately people do not have this knowledge or avoid using them because they are afraid of complexity. This approach can help building these models from simple feature models and software architecture.

In this project we used Queuing Networks as a follow up to our other project (CS700 course) where we used bare simulations to overcome the problem. The results we obtained from the QN here are consistent with what we got in our simulations. But, these results are easier and faster to obtain; for simulation we need to run the simulation for a long time to get in steady state and do so much analysis to remove unimportant data. It seems that starting with the simulation and following up with QNs is a good approach and reduces the hassle of dealing with simulation data later on.

We can also keep the models constantly consistent with the monitoring data obtained from the real system to use them for planning purposes. But this seems to be a different branch of our research which we have not dealt with yet. It seems that when certain information are available (e.g., service demands) using Queuing Networks and in general analytical models are easier way to get the same result with less overhead.

Since we already had the result of simulation, we used that to initiate our analysis. We did not use “Two-Level Iterative Queuing Modeling of Software Contention”, but, it seems that we may be able to use it instead of simulation to start the process. However, more research in this area is needed and it demand future research maybe as future works.

Acknowledgement

Special thanks to Ahmed Elkhodary for providing the feature modeling language in GME. This work is a part of bigger research which its preliminary results [7] are published. We reused some parts the report we presented in CS700 course here; namely the three first pages which are dealing with the introduction are as same as the CS700 report with minor changes to fit the purpose of this project.

References

[1] "expedia," *http://www.expedia.com/*.

[2] priceline, "priceline," *http://www.priceline.com/*.

[3] N. Medvidovic and R.N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Trans. Softw. Eng.*, vol. 26, 2000, pp. 70-93.

[4] G. Edwards, S. Malek, and N. Medvidovic, "Scenario-Driven Dynamic Analysis of Distributed Architectures," *LNCS*, vol. 4422, 2007, p. 125.

[5] GME, "GME," *http://www.isis.vanderbilt.edu/Projects/gme/*.

[6] G.J. Kiczales, J.O. Lamping, C.V. Lopes, J.J. Hugunin, E.A. Hilsdale, and C. Boyapati, *Aspect-oriented programming*, Google Patents, 2002.

[7] A. Elkhodary, S. Malek, and N. Esfahani, "On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems," *4th International Workshop on models@runtime*, Denver, Colorado, USA: 2009.

[8] D.A. Menasce, L.W. Dowdy, and V.A. Almeida, *Performance by Design: Computer Capacity Planning By Example*, Prentice Hall PTR, 2004.