Ranking Different Software Architectures Based on QoS

CS700 - Final Project Report

Naeem Esfahani

May 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 systems.

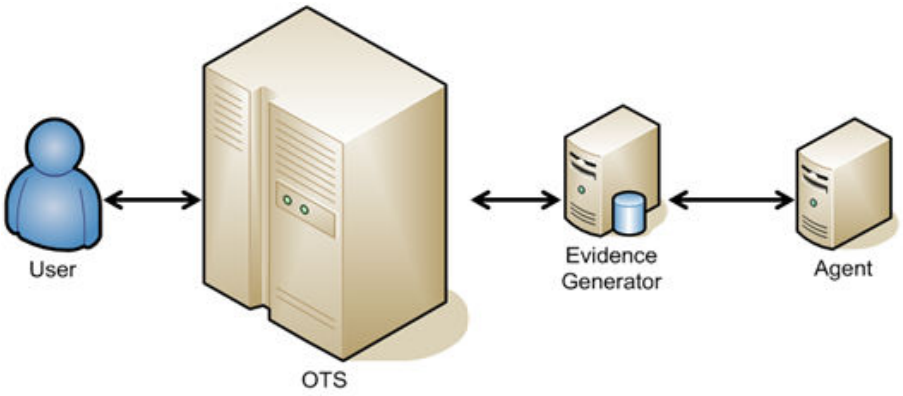

Figure 1. High level view of Online Ticketing Systems

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.

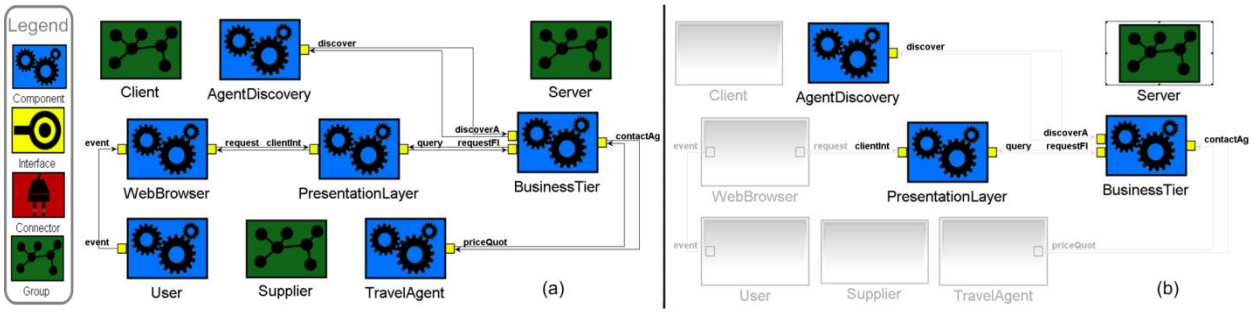

Figure 2. a) Core architectural model for OTS, and b) Server group is selected.

The architectural model depicted in Figure 2, is generated with 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].

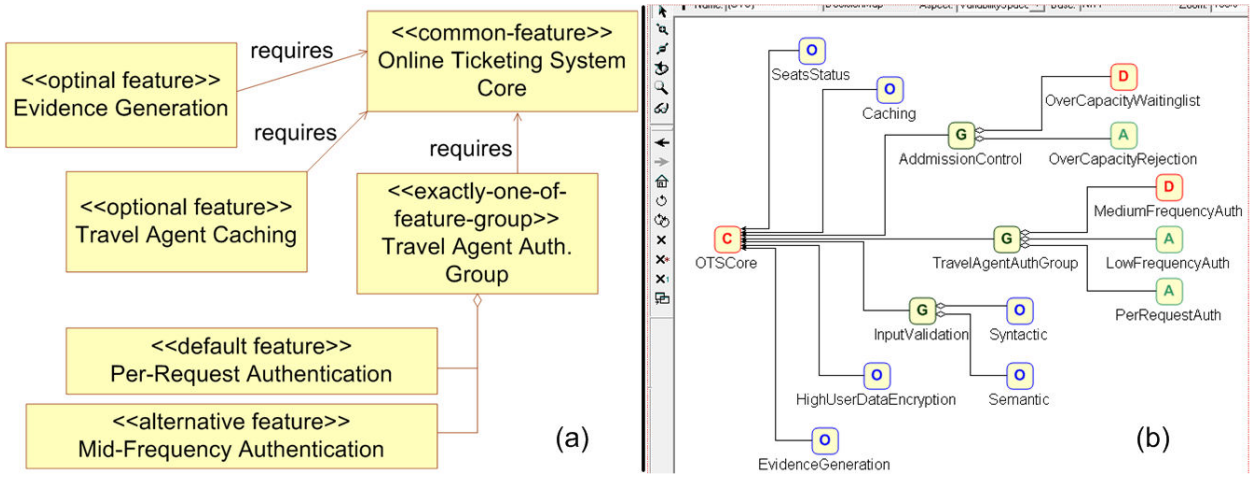

Figure 3. Feature Models. a) abstract feature model, and b) feature model implemented as DSML in GME

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. FFigure 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.

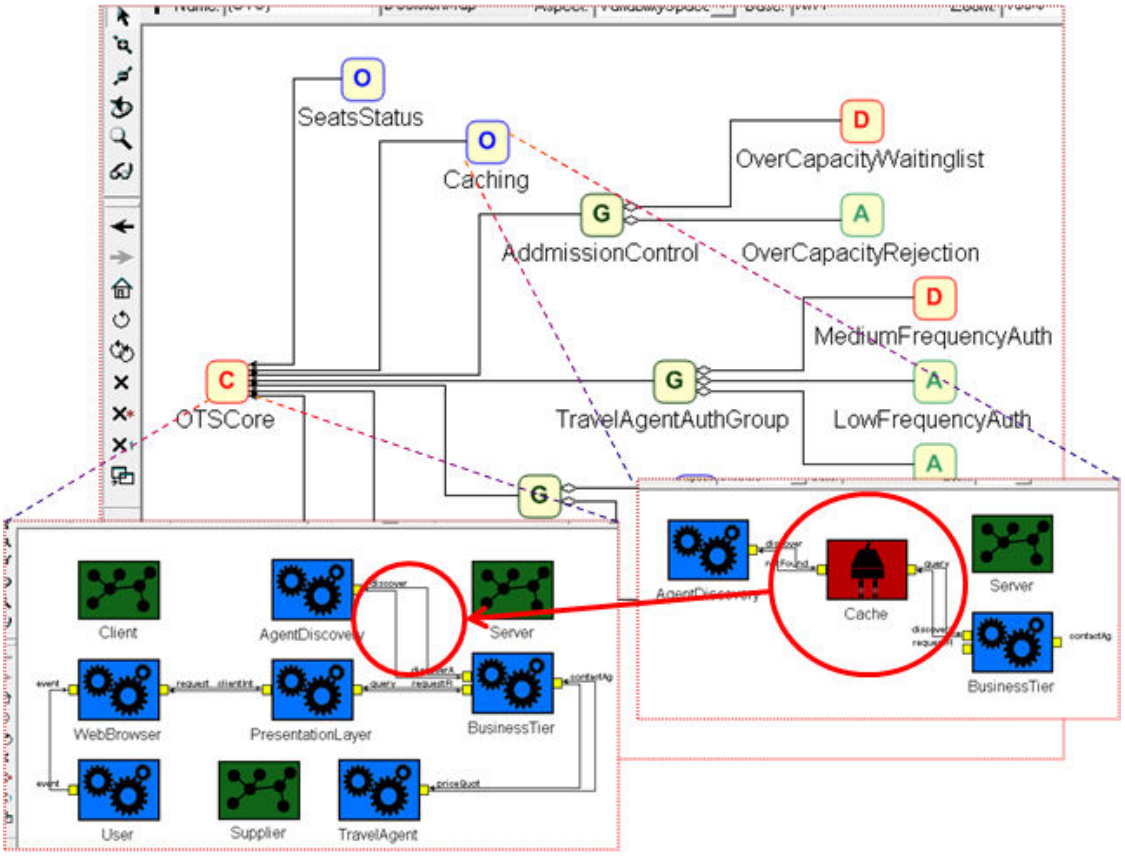

Figure 4. Caching feature's architecture is weaved into the Core architecture.

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.

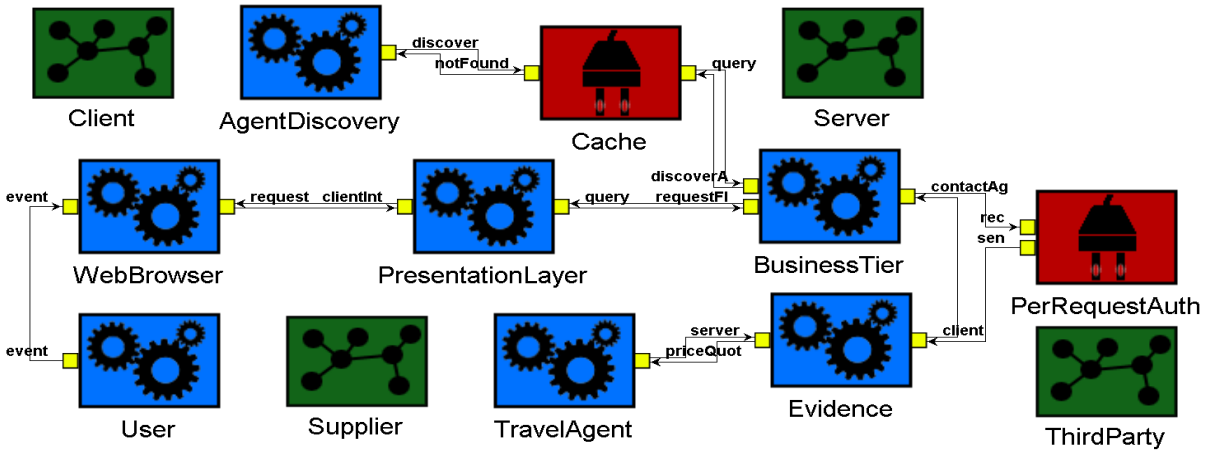

Figure 5. Architectural combination as a result of selecting Core, Caching, Evidence Generation and Per Requeset Authentication features

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 based on the simulation 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. 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 during simulation in design time. Different architectural configurations are built by combining different features (recall Section 1.3). The response time is measured on the WebBrowser's output interface (*request* in Figure 5).

3 Methodology Used

We captured different feature combinations as different level for a factor. The best approach is to have a factor corresponding to a feature getting two values (on/off). This would give us a 2^k Factorial experiment which can easily get out of hand and cause space explosion. Moreover, the tool support for more than two factors is very limited. Therefore, we encoded the problem as one factor experiment.

The experiment is broken down into two phase: significance test, interaction test. In significance test, the features are selected exclusively. Based on early inspections; the prominent features which have the larger effect on the response time are discovered. In interaction test the interactions of prominent features are included in the experiment as well and the main simulation is done for both exclusive feature selections and interaction of new features.

Figure 6 shows this design for OTS. In the case of OTS, the prominent features are Evidence generation, Caching and Per Request Authentication. This case study did not have many features, however when the number of features are very much this can help in pruning the problem space very much.

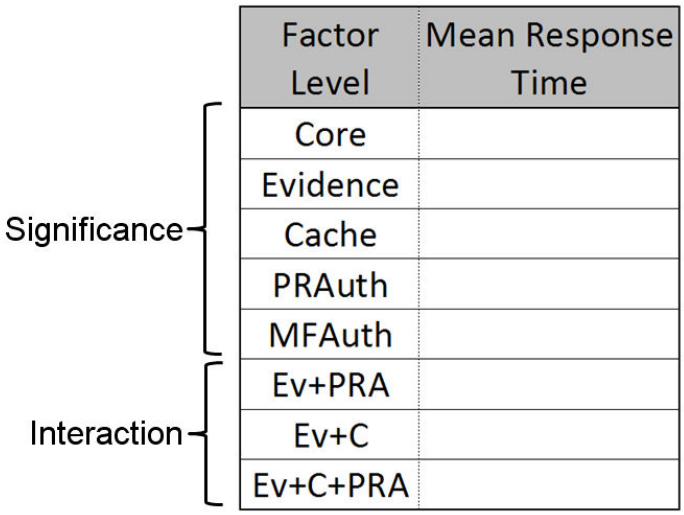

Figure 6. Two phase experiment design

We did a simulation for each level of the factor (i.e., feature selection). The simulation was conducted with XTeam and based on the generated architectural models. The response time was measured from the output interface of WebBrowser. For transient elimination the Batch method was used. For computing the mean and confidence intervals we used the Batch method and its stopping criteria. This result helped us to rank different architectures.

To test this result we conducted a one factor ANOVA. By taking ten samples of response time after the transient phase, we build a one factor ANOVA. The result of our experiment is presented in following sub-sections.

3.1 Simulation

XTeam is a simulation framework for architectural models. Based on different feature selections we build XTeam models and simulated them. XTeam is based on discrete event simulation and has a logical clock. The response time is measured by number of logical clocks elapsed.

The inter-arrival time of input event which is generated by the user is according to exponential distribution. The lambda parameter for the exponential distribution is set to put the system under stress. However, this stress is not that much to put the system in the thrashing state. As mentioned before we did a preliminary analysis to find the most prominent features. We selected a lambda to put the feature selection with the highest response time near to thrashing. So, we are sure that the other feature selections do not go into thrashing state and the simulation result is as expected.

Figure 7 shows variance by batch size for each feature combination. From the results it is obvious that after 200 samples the system gets into the steady state for all of them. Therefore we removed the first batch (i.e., the first 200 sample of response time). Note that our simulation has

generated many numbers and therefore most of preconditions for doing ANOVA and computing confidence intervals are met.

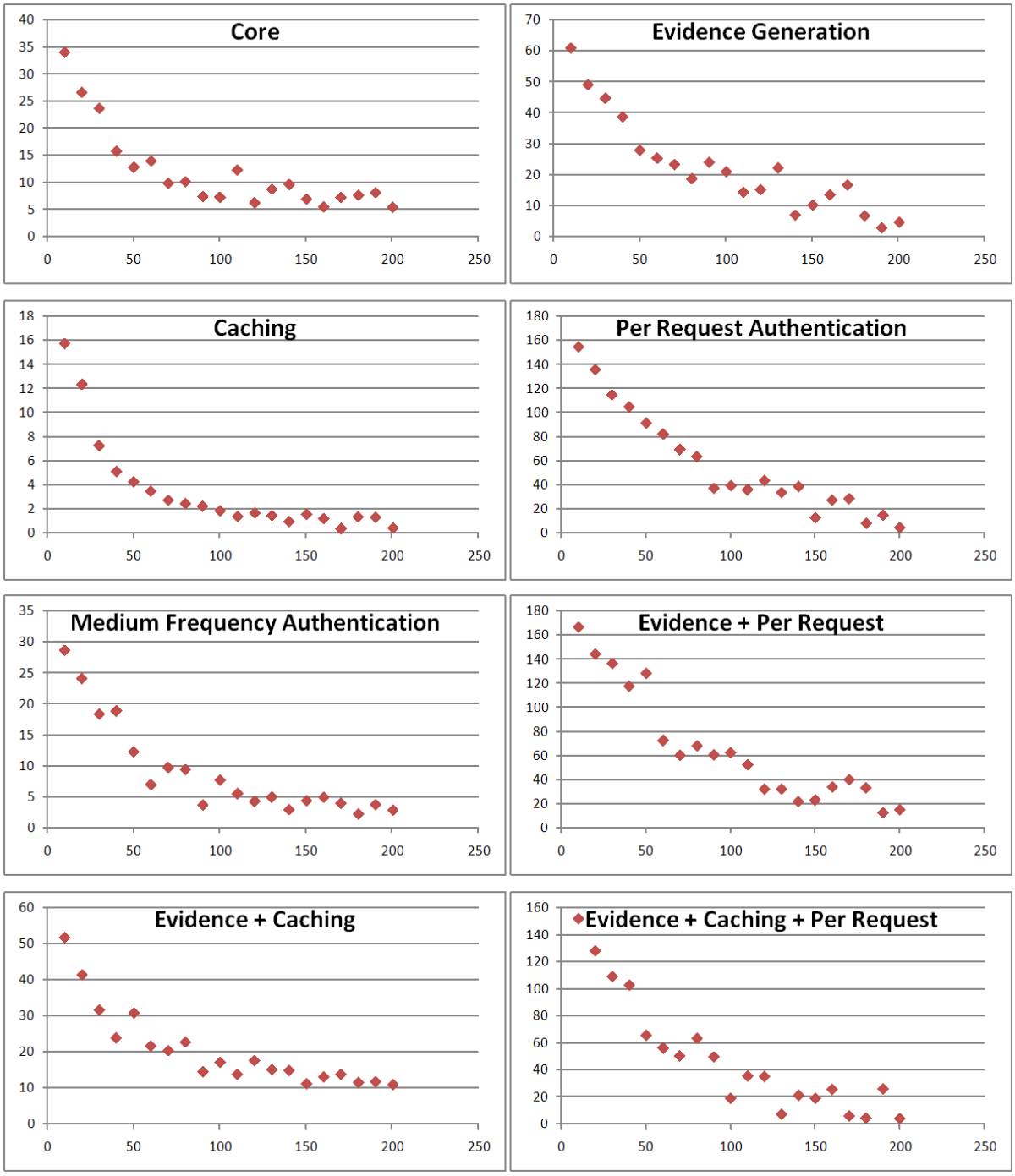

Figure 7. Transient Elimination using Batch method

The mean and confidence intervals computed (using Batch method, according to the stopping criteria) are shown in Figure 8. To have a simple intuition we included two more columns besides the calculated result; under Actual Effect we subtracted the response time for

feature combinations from the response time of core feature. Using this numbers we have made a simple guess when we have a combination of the features just by adding the effects together. In the main project (which this project is part of that) the goal is to come up with more advanced ways to this kind of analysis using data mining.

Here, we can see for the combination of Evidence Generation and Per Request Authentication we have a good guess. However, for the other cases where caching is included the guess is not that perfect. This is due to the fact that the Caching takes resources from the system and when the system is under more load (having more concurrent requests because of authentication and evidence generation). The load of caching affects the system and avoids it from having the pure impact.

	Factor (**Features Combination**)				Response (**Metric**)				
	Evidence Generation	Caching	Per-Request Auth.	Medium Frequency Auth.	Response Time				
					Mean	ConfLow 95%	ConfUp 95%	**Actual Effect**	
Significance Test	0	0	0	0	24.3819	23.924	24.8397		
	1	0	0	0	34.49	33.8828	35.0972	10.10813	
	0	**1**	0	0	17.5291	17.1652	17.893	-6.85282	
	0	0	**1**	0	34.1866	33.3295	35.0437	9.804689	**Simple**
	0	0	0	**1**	25.4765	25.0455	25.9075	1.09461	**Guess**
Interactions Test	**1**	0	**1**	0	43.7517	42.7886	44.7147	19.36976	19.913
	1	**1**	0	0	32.7967	32.2171	33.3764	8.414843	3.2553
	1	**1**	**1**	0	40.8916	39.9285	41.8546	16.50966	13.06

Figure 8. Mean and confidence intervals computed using batch method

Figure 9 shows the confidence interval for the response time according to feature selection. It shows that the effect of the Evidence Generation on response time is almost the same as Per Request Authentication. Moreover, the effect of Medium Frequency Authentication on response time is barely different from the Core's effect.

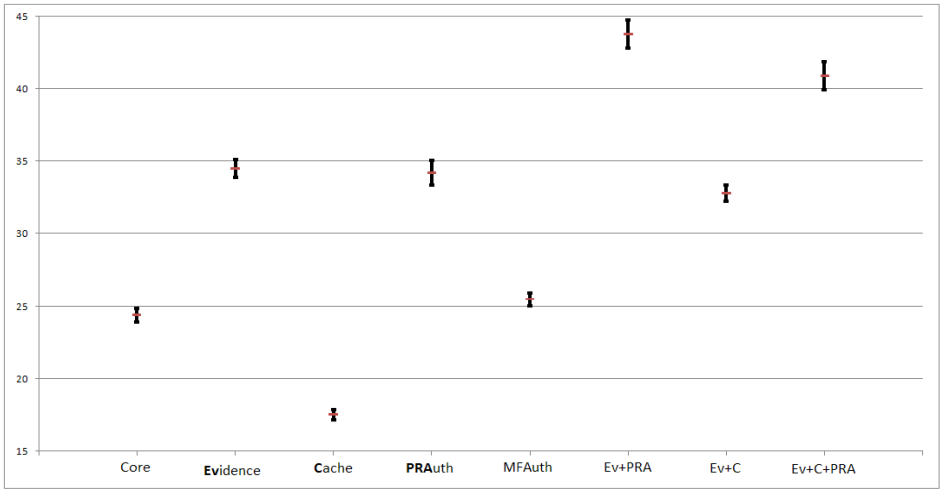

Figure 9. Confidence intervals for mean of response time in different configurations

3.2 ANOVA

As another test, we did a one factor ANOVA to see if the mean of response time for different feature combination is the same. After removing the transient part of simulation we took ten random samples of response time for each feature combination. Figure 10 shows the data used for ANOVA.

Core	Evidence	Cache	PRAuth	MFAuth	Ev+PRA	Ev+Cache	Ev+C+PRA
21.51113	30.94172	20.9517	30.33639	28.70901	39.89597	30.18391	33.70322
20.5829	35.32981	16.07803	37.01343	26.83819	39.39868	31.92519	45.4843
21.77108	33.3296	17.03116	31.21694	24.60643	59.55698	32.34826	30.15467
25.86763	30.19304	17.87598	40.02986	22.35055	49.5676	35.06328	46.64558
29.1959	49.60813	18.64181	26.25097	32.08715	36.86989	36.38723	46.56037
30.53711	35.88266	16.64817	29.46692	23.63155	49.35754	31.4821	34.40236
24.44325	33.14643	17.69031	40.52027	27.17258	32.20139	37.60165	39.68256
24.59928	28.2936	17.3138	46.02936	22.26455	49.54168	34.70651	48.46248
23.43006	35.58542	17.60207	30.59231	23.88995	42.03592	30.9783	47.33173
21.88065	32.58989	15.45781	30.40943	23.21512	39.09091	27.29099	36.48828

Figure 10. Raw data for doing ANOVA

By a glimpse one can say the mean of response time for different feature combinations is not the same and therefore the Null-hypothesis will be strongly rejected. We can see the same fact from the previous test (Figure 9) as well.

Groups	*Count*	*Sum*	*Average*	*Variance*
0000	10	243.819	24.3819	11.108
1000	10	344.9003	34.49003	34.29047
0100	10	175.2908	17.52908	2.285568
0010	10	341.8659	34.18659	39.84219
0001	10	254.7651	25.47651	10.08453
1010	10	437.5165	43.75165	65.3147
1100	10	327.9674	32.79674	9.766567
1110	10	408.9155	40.89155	46.11613

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*	*Reject H0*
Between Groups	5379.975	7	768.5678	28.10015	3.18E-18	2.139656	**TRUE**
Within Groups	1969.273	72	27.35102				
Total	7349.248	79					

Figure 11. The result of ANOVA in Microsoft Excel

As depicted in Figure 11, and according to our expectation the Null-hypothesis is strongly rejected. After getting this result, using Turkey-Kramer method we tested to see which feature combinations are significantly different. The result for this test is shown in Figure 12. As we can see we got the same result that we had already obtained from the Batch mean and confidence interval computation.

	Core	Evidence	Cache	PRAuth	MFAuth	Ev+PRA	Ev+C	E+C+PRA
Core	FALSE	TRUE	TRUE	TRUE	**FALSE**	TRUE	TRUE	TRUE
Evidence	TRUE	FALSE	TRUE	**FALSE**	TRUE	TRUE	TRUE	TRUE
Cache	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
PRAuth	TRUE	**FALSE**	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
MFAuth	**FALSE**	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
Ev+PRA	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE
Ev+C	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
E+C+PRA	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE

Figure 12. Computing the feature combinations which are significantly different using Turkey-Kramer method.

In the intersection of each row and column in Figure 12, the Turkey-Kramer method has been applied to test if the two features are significantly different. TRUE means that they are distinguishable and FALSE means that they are not that different from each other.

4 Analysis of Results

We used two statistical methods to compare generated software architectures for different feature combinations. The analysis showed us that these feature combinations are not acting similar to each other regarding response time.

In the first method, we did this comparison using confidence intervals and we realized that Evidence Generation and Per Request Authentication have the same result on the response time because their confidence intervals have intersections containing each other's mean. Moreover, we realized that the Medium Frequency Authentication does not have much impact on the response time since its confidence interval is barely different from Core's confidence interval.

In the second method, we used ANOVA to see if the mean of response time for different feature combinations is different from the other ones. The Null-hypothesis was rejected and using Turkey-Kramer method we got the same result as the first method. It seems that both the two methods declaring the same fact in different ways.

Moreover, based on what we learned for features independently we assembled a simple guess to predict the effect of selecting the features together. The guess turned out to be good for the cases that the response time increases by features. For the features which decrease the response time the load should be considered as well. This load will have interaction with other features and avoid the feature to have an independent and direct impact.

5 Concluding Remarks

In our method, we decreased the amount of input that user needs to enter into the system. The designer can model the features in design time and provide the simulation with expected effect of the features independently and get the results for all the combinations. The models can be kept during the runtime. The information should be updated to enable the user of the system to have an estimate about the role of each feature on the non-functional requirements. This knowledge can be used in emergency situations when someone is willing to reduce the features of the system to keep it in working condition.

We got the result that we were seeking in the context of this project. If we could manage to conduct a full factorial experiment design using available tools we could get more interesting results by doing more than ranking that we have in Figure 9. We could extract feature interactions and do some kind of learning. However, since the feature space tends to get very enormous, we may not be able to rely on pure statistical methods. For this kind of analysis we may be able to use data mining techniques which are having roots in statistics.

Acknowledgement

Special thanks to Ahmed Elkhodary for providing the feature modeling language in GME.

References

[1] "expedia," *http://www.expedia.com/*.
[2] "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," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4422, 2007, p. 125.
[5] ISIS, Vanderbilt University, "Generic Modeling Environment," *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.