Alternative Design using an Example

We have now discussed the two key elements of a C&C architecture view of a software system, and how they work together. Let us now discuss an example, putting these concepts together.

Suppose we have to design and build a simple system for taking an on-line survey of students on a campus. There is a set of multiple choice questions, and the proposed system will provide the survey form to the student, who can fill and submit it on-line. We also want that when the user submits the form, he/she is also shown the current result of the survey, that is, what percentage of students so far have filled which options for the different questions.

The system is best built using the Web; this is the likely choice of any developer. For this simple system, traditional 3-tier architecture is proposed. It consists of a client which will display the form that the student can fill and submit, and will also display the results. The second component is the server, which processes the data submitted by the student, and saves it on the database, which is the third component. The server also queries the database to get the outcome of the survey and sends the results in proper format (HTML) back to the client, which then displays the result. A figure giving the C&C view is shown in Figure 1. Note that the client, server, and the database are all different types of components, and hence are shown using different symbols. Note also that the connectors between the components are also of different types. The diagram makes the different types clear, making the diagram stand alone and easy to comprehend. Note that at the architecture level, a host of details are not discussed.

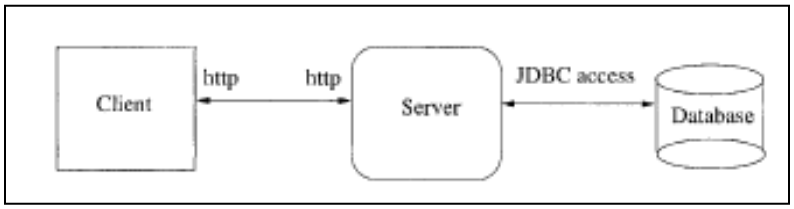

Figure 1 : Architecture of Survey System

How is the URL of the survey set? What are the modules that go in building these components and what language they are written in? Questions like these are not the issues at this level.

Note also that the connector between the client and the server explicitly says that http is to be used. And the diagram also says that it is a Web client. This implies that it is assumed that there will be a Web browser running on the machines from which the student will take the survey. Having the http as the connector also implies that there is a proper http server running, and that the server of this system will be suitably attached to it to allow access by clients. In other words, the entire infrastructure of browser and the http server, for the purposes of this application, mainly provides the connector between the client and the server (and a virtual machine to run the client of the application).

There are some implications of choice of this connector on the components. The client will have to be written in a manner that it can send the request using http (this will imply using some type of scripting language or HTML forms). Similarly, it also implies that the server has to take its request from the http server in the format specified by the http protocol. Furthermore, the server has to send its results back to the client in the HTML format. These are all constraints on implementing this architecture. Hence, when discussing it and finally accepting it, the implications for the infrastructure as well as the

implementation should be fully understood and actions should be taken to make sure that these assumptions are valid.

Extension I

The above architecture has no security and a student can take the survey as many times as he wishes. Furthermore, even a non-student can take the survey. Now the Dean of students wants that this system be open only to registered students, and that each student is allowed to take the survey at most once. To identify the students, it was explained that each student has an account, and their account information is available from the main proxy server of the institute.

Now the architecture will have to be quite different. The proposed architecture now has a separate login form for the user, and a separate server component which does the validation. For validation, it goes to the proxy for checking if the login and password provided are valid. If so, the server returns a cookie to the client (which stores it as per the cookie protocol). When the student fills the survey form, the cookie information validates the user, and the server checks if this student has already filled the survey. The architecture for this system is shown in Figure 2. Note that even though we are saying that the connection between the client and the server is that of http, it is somewhat different from the connection in the earlier architecture. In the first architecture, plain http is sufficient. In this one, as cookies are also needed, the connector is really http + cookies. So, if the user disables cookies, the required connector is not available and this architecture will not work.

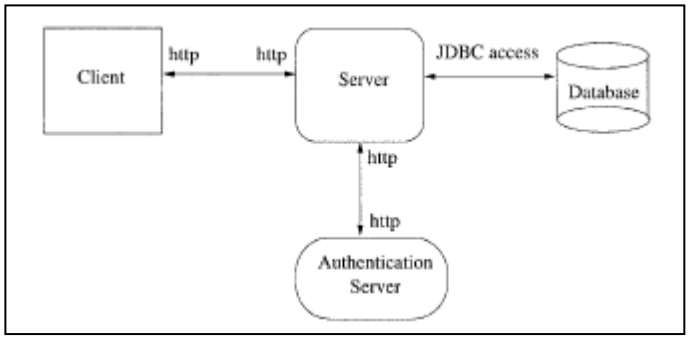

Figure 2 : Architecture of Survey System with Authentication

Extension II

Now suppose, we want the system to be extended in a different way. It was found that the database server is somewhat untenable, and is frequently down. It was also felt that when the student is given the result of the survey when he submits the form, a somewhat outdated result is acceptable, as the results are really statistical data and a little inaccuracy will not matter. We assume that the survey result can be outdated by about 5 data points (even if it does not include data of 5 surveys, it is OK). What the Dean wanted was to make the system more reliable, and provide some facility for filling the survey even when the database is down.

To make the system more reliable, the following strategy was thought. When the student submits the survey, the server interacts with the database as before. The results of the survey, however, are also stored in the cache by the server. If the database is down or unavailable, the survey data is stored locally in a cache component, and the result saved in the cache component is used to provide the result to the

student. (This can be done for up to 5 requests, after which the survey cannot be filled.) So, now we have another component in the server called the cache manager. And there is a connection between the server and this new component of the call/return type. This architecture is shown in Figure 3.

It should be clear that by using the cache, the availability of the system is improved. The cache will also have an impact on performance. These extensions show how architecture affects both availability and performance, and how properly selecting or tuning the architecture can help meet the quality goals (or just improve the quality of the system). (Of course, detail level decisions like how a particular module is implemented also has implications on performance, but they are quite distinct and orthogonal to the architecture-level decisions.) We will later do a formal evaluation of these different architectures to see the impact of architectural decisions on some quality attributes.

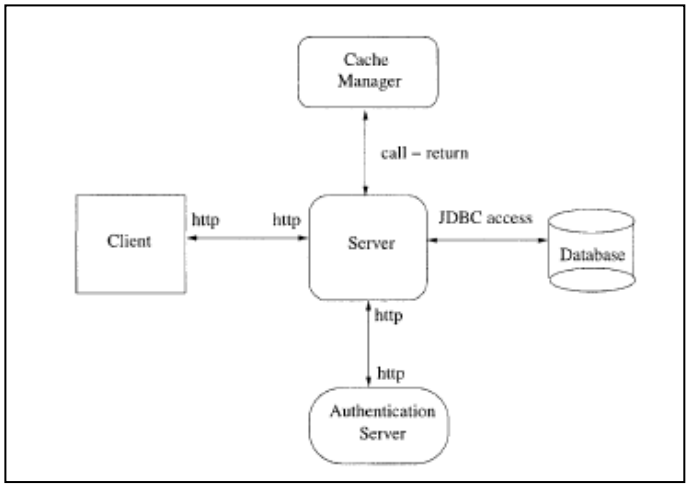

Figure 3 : Architecture of Survey System with Cache

Software architecture is an evolving area, and there are many issues that have not been fully resolved. In this section we discuss some issues in order to further clarify concepts relating to architecture and designing of architectures.

Evaluating Architectures

Architecture of a software system impacts some of the key nonfunctional quality attributes like modifiability, performance, reliability, portability, etc. The architecture has a much more significant impact on some of these properties than the design and coding choices. That is, even though choice of algorithms, data structures, etc., are important for many of these attributes, often they have less of an impact than the architectural choices. Clearly then evaluating a proposed architecture for these properties can have a beneficial impact on the project—any architectural changes that are required to meet the desired goals for these attributes can be done during the architecture design itself.

There are many nonfunctional quality attributes. Not all of them are affected by architecture significantly. Some of the attributes on which architecture has a significant impact are performance, reliability and availability, security (some aspects of it), modifiability, reusability, and portability. Attributes like usability are only mildly affected by architecture.

How should a proposed architecture be evaluated for these attributes? For some attributes like performance and reliability, it is possible to build formal models using techniques like queuing networks and use them for assessing the value of the attribute. However, these models require information beyond the architecture description, generally in forms of execution times, and reliability of each component.

Another approach is procedural—a sequence of steps is followed to subjectively evaluate the impact of the architecture on some of the attributes. One such informal analysis approach that is often used is as follows. First identify the attributes of interest for which architecture should be evaluated.

These attributes are usually determined from stakeholder's interests— the attributes the different stakeholders are most interested in. These attributes are then listed in a table. Then for each attribute, an experience based, subjective analysis is done (though quantitative analysis can also be done), to assess the level supported by the architecture. The analysis might mention the level for each attribute (e.g., good, average, poor), or might simply mention whether it is satisfactory or not. Based on the outcome of this analysis, the architecture is either accepted or rejected. If rejected, it may be enhanced to improve the performance for the attribute for which the proposed architecture was unsatisfactory. Many techniques have been proposed for evaluation, and a survey of them is given in [51]. Here we briefly discuss some aspects of a more elaborate and formal technique called architectural tradeoff analysis method.

The ATAM Analysis Method

The architectural tradeoff analysis method (ATAM) [105, 35], besides analyzing the architecture for a set of properties, also helps in identifying dependencies between competing properties and perform a tradeoff analysis. We will, however, mostly focus on the analysis. The basic ATAM analysis has the following steps, which can be repeated, if needed :

1. *Collect Scenarios.* Scenarios describe an interaction of the system. For architecture analysis, scenarios list the situations the system could be in and for which we would like to evaluate the architecture for different attributes. Besides normal scenarios, exceptional scenarios of interest should also be mentioned.
2. *Collect Requirements or Constraints.* These specify what are the requirements for the system. That is, what is expected from the system in these scenarios. The scenarios together with requirements form the basis for evaluation—we want to ensure that the software will satisfy these requirements in these scenarios. These requirements essentially specify the desired levels (hopefully quantitatively) for the quality attributes of interest. So, for example, instead of saying performance is of interest in this system, a constraint or a requirement will be like "the average response time should be less than 1 ms."

3. *Describe Architectural Views.* The views of different proposed architectures are collected here. These are the architectures that will be evaluated. What views are needed to describe a proposed architecture depends on what analysis needs to be performed, which is driven by the requirements or constraints. We will limit our attention to component and connector view only.
4. *Attribute-Specific Analysis.* Now we have the proposed architectures, the different quality attributes of interest for the system, and the different scenarios under which these attributes should be evaluated. In this step, each quality attribute is analyzed separately and individually. The analysis should result in what levels an architecture can support for the quality attribute. So, for example, an analysis can result in a statement like "the availability of this system is 0.95." Once the analysis for all the attributes is done, it can be seen to what degree the requirements identified earlier are met. The outcome of this analysis can become the basis for selecting one architecture over other. It can also form the basis of changing a proposed architecture in an attempt to meet the desired levels. If an architecture is changed, then the whole analysis needs to be repeated, making ATAM a spiral process.
5. *Indentify Sensitivities and Tradeoffs,* Prom the analysis, for each attribute, sensitivity of the different elements in the architecture view should be determined. That is, how much impact does an element have on the attribute value. Prom this we identify the *sensitivity points,* which are the elements that have the most significant impact on the attribute value. Sensitivity points are these elements whose change will have the maximum impact on the quality attribute. *Tradeoff points* are those elements that are sensitivity points for multiple attributes. Changing these elements will have a significant impact on multiple attributes. These elements are where tradeoffs decisions will have to be taken, particularly if changing it favorably affects one attribute but negatively affects the other. For example, in a n-tier architecture, a server will be the tradeoff point as it significantly affects performance as well as availability. Replicating the server and performing updates on each server (to keep each current) can improve the availability, but can have a detrimental effect on update performance.

An Example

Earlier in the chapter, we gave an example of the student-survey system. We will take that example for evaluation and consider the second and the third architectures proposed (we do not consider the first one as it does not have the same functionality as others). For analysis, we add another architecture, in which the cache component is between the server and the database component. That is, in this architecture, each request is directly sent to the cache, which then decides whether to respond using data from the cache or from the database. This architecture is shown in Figure 4. We assume that the cache component in these architectures updates the database

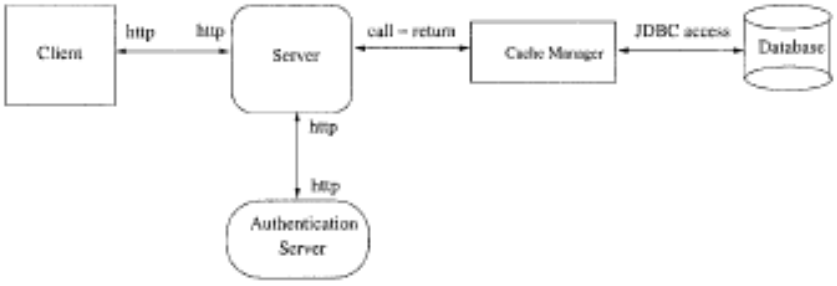

Figure 4 : Another architecture for the student survey system

as well as its own data after every 5th survey. (For analysis we add a requirement on data currency to allow this. The requirement is given below.) We focus only on survey-taking and ignore the feature of just getting the survey result. Let us now analyze these architectures using ATAM. First we hst the scenarios of interest. These are:

• *SI. A* student submits the survey form and gets current results of the survey (normal scenario; all servers are up, load normal.)
• *S2.* A student tries to take the survey many times.
• *S3.* The database server is temporarily down.
• *S4'* The network/system is highly loaded.

Some of the requirements or constraints for the system are:

• *Security. A* student should be allowed to take the survey at most once.
• *Response Time. A* student should get a response time of less than 2 sec on an average, 80% of the time.
• *Availability.* The system should at least have availability of 0.85 (that is, when a student comes, there is 85% chance that he can successfully take the survey).
• *Data Currency.* The survey result given to a student should be reasonably current and should not be older than 5 submissions before.

Now let us evaluate the three architecture proposals for these attributes. For analysis, we will look at each of the attributes and then study the three architectures under the scenarios relevant for that attribute. For security and data currency, we will analyze based on our understanding of the architecture. For availability and response time, formal models are possible. We will use simple probability-based approach here. For the availability analysis, we assume that the cache is on the same machine as the server, the database is on a different machine, and that availabihty of each of the machines is 0.9. We also assume that when the database goes down, during its repair time, on an average, 10 student survey requests come. For response time, we assume the following response times:

Component	Normal conditions	Heavily loaded
Server + security	300ms	600ms
Database	800ms	1600ms
Cache	50ms	50ms

We also assume that a timeout of about 2 seconds is used when the server tries to access the database, and that the network is heavily loaded 1% of the time. We do not show all the computations here, but illustrate a few. The availability for the first architecture is the probability that both the server and database are up, as that is the only case in which a request can be serviced. This is 0.9 * 0.9 = 0.81. For the second and the third case, it is slightly more complex. When the database is down (on an average for 10 requests), up to 5 requests can still be serviced. Hence, even when the database is down, the system is up for half the time. This gives us an additional availability of 0.5 * 0.9 * 0.1 = 0.045 (half the probability that server is up and database is down). So, the availability of these architectures is 0.81 + 0.045 = 0.855. Determining the average response time is slightly more involved, as we have to consider both the heavy load and normal load situations. In normal load, for first architecture, the average response time will be 300 -j- 800 = 1100 ms. When the database is down, then this architecture cannot service a request. For the

second architecture, the response time is 300 + 800 + 50 = 1150 ms in the normal scenario. When the database is down, some requests can be serviced (probability computed above) by the cache but cache is used only after the database times out. That is, the response time in this scenario is 300 + 2000 + 50 = 2350 ms. For the third architecture, in the normal scenario, the average response time is 350 * 0.8 (for those requests serviced

Property (Scenario)	Architecture 1	Architecture 2	Architecture 3
Security (S1)	Yes	Yes	Yes
Security (S2)	Yes	Yes	Yes
Response time (S1)	1100	1150	550
Response time (S3)	N/A	2350	550
Response time (S4)	2200	2300	1100
Availability (S3)	0.81	0.855	0.855
Data Currency (S1)	Yes	Yes	Yes
Data Currency (S2)	N/A	Yes	Yes

Table 5 : Analysis of the architecture options. by cache)

plus 1350 * 0.2 (for those that go to the database). That is, the average response time is 550 ms. When the database is down, the response time for the requests that can be serviced remains the same as they are serviced from the cache in a normal manner. Similar analysis can be done when the network is congested. For simplicity, we will double these times (as the response time when the system is congested is double that when it is not). With these, we build a table for the different attributes for the scenarios of interest. This is given in Table 5. Prom this table we can clearly see that security and data currency requirements are satisfied by all three architecture options. As the probability of normal scenario is over 0.8 for all architectures, and response time in normal scenario is less than the required for all, the response time requirement is also met by all the three architectures. However, the availability requirement is met by only the second and the third architectures. Of these two architectures, the third should be preferred as it provides a better response time.