India

H3: Beer Web Store case

Marjus Nielsen

Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark

Group: India

20064684

marjus@daimi.au.dk

17. 03. 2008

H3 – 1.1

Outline how you would approach the tasks of creating the architecture for the Beer Web Store. Consider, e.g., which steps would you take and in which order?

I would first do a brainstorm to list possible issues relevant to the system. Thereafter i would use the brainstorm as basis for making a list of important quality attribute scenarios written on the form proposed by [Bass et al. 2007]. The quality attributes in the list should then be prioritised, especially if some of them are non-orthogonal to the others. This is all done in order to reveal non-functional requirements. diskutér/inddrag evt. også ABC

The quality attribute scenarios should then be used as input for attribute-driven design where I iteratively choose a module or a part of the system to decompose, and then refine that model according to the architectural drivers found via the quality attributes. Architectural drivers are requirements with high priority that have an effect on the system architecture.

The above steps will result in a architectural description which is best described in different views. I will use the proposed viewpoints from [Christensen et al., 2004], Module, Component and connector and an Allocation viewpoint.

Give concrete examples of what elements, relations, and structures as defined in [Bass et al., 2007] could be in relation to an architecture for the Beer Web Store.

In the module view in Figure 1 the packages are elements and the dependencies between the packages are relations. The module view itself is the structure.

struktur: hvad der faktisk er i systemet + view: representation af struktur

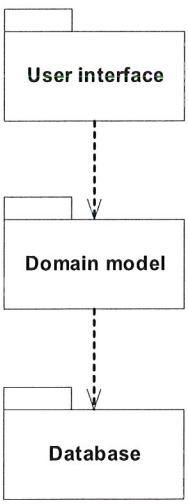

Figure 1 Top level module view

In Figure 2 one of the modules is decomposed into 4 submodules. A uses structure shows which modules are used by other modules.

The module view is a compile-time view with emphasis on units of implementation and how they relate to eachother.

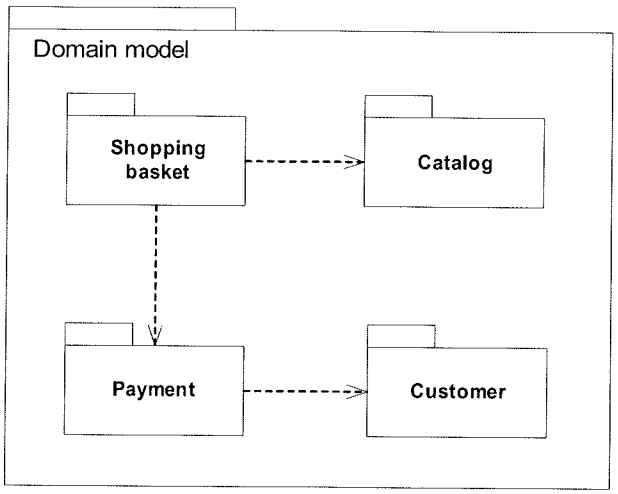

Figure 2 Decomposed module view showing the Domain Model

The component and connector view in Figure 3 Component and connector view shows the runtime structure of the system. The components and connectors are elements and the protocols are relations.

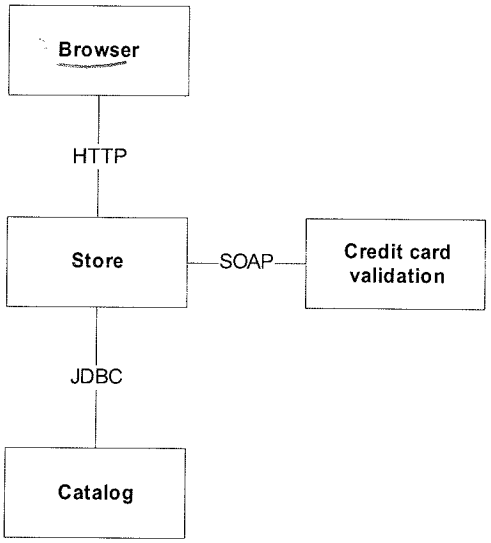

Figure 3 Component and connector view

Figure 4 shows an allocation structure and depicts how software elements are related to hardware, ie. which software elements are deployed to what hardware units. Communication paths between the hardware units are also shown. The deployment view can help estimating performance and thus could be enriched with properties about the hardware.

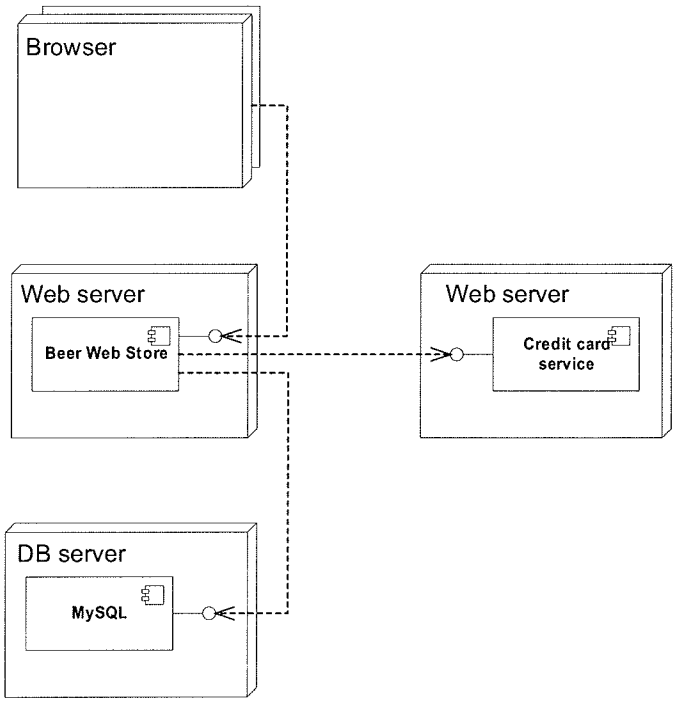

Figure 4 Allocation view

Apply [Perry and Wolf, 1992]'s model of software architecutre as

Software architecture = {Elements, Form, Rationale}

to the Beer Web System.

[Perry and Wolf, 1992] define software architecture as a combination of elements, their form and a rationale for choices made. There are 3 kinds of elements: processing elements, data elements and connecting elements.

Processing elements perform transformations on data elements. Transformation of data elements is e.g. loading a file into memory or performing a calculation.

Data elements contain information that can be transformed by processing elements. Data elements can e.g. be files on disk and objects in memory.

Connecting elements glue the pieces of the architecture (method calls, shared resources). A conneting element transfers data elements without transforming the data. A connecting element can be a processing element or a data element or both.

[Perry and Wolf, 1992]'s definition of software architecture is mostly focusing on the architecture at runtime. Compared to [Bass et al., 2007] the processing elements in [Perry and Wolf, 1992] would be things such as the store and the catalog shown in Figure 3. Data elements would be the JDBC, HTTP and SOAP messeges that are passed between the processing elements.

The form consists of properties and relationships. The properties constrain the choices of architectural elements. Relationships constrain how elements may interact and how the elements are organized.

The rationale is according to [Perry and Wolf, 1992] an underlying but integral part of the architecture. Choices made and alternative solutions are recorded in the rationale.

For the Beer Web Site the rationale would be arguments about why the architect has chosen a layered architecture, why he uses a Facade pattern, and possibly even considerations about why a web site was chosen instead of e.g. a thin client application.

It can be noted that other architects e.g. [Fielding 2000] think that the rationale isn't really part of the architecture, in the same way the blueprints aren't part of the architecture of building architecture. The building remains in the same condition if the blueprints are disposed.

Reflect on what happens if the words "software" and "computing" are removed from the definition:

Bass et al definition:

"The [software] architecture of a program or [computing] system is the structure or structures of the system, which comprise [software] elements, the externally visible properties of those elements, and the relationships among them."

Without the words "software" and "computing" the definition of architecture becomes much more general and can be used as a definition of other types of architecture such as traditional (building) architecture, network architecture and hardware architecture.

[Perry and Wolf] compare software architecture to other types of architecture. For building architecture they argue that you need different structures (views) to describe the same building. One for view or diagram for each particular aspect relevant to the reader of the diagrams. This can be floor-plans, electrical diagrams, air conditioning etc.

[Alexander et al. 1977] says about building architecture patterns:

> *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."*

and this fits very well with the purpose of software architectural patterns, where effort has been laid in creating patterns for solving the same problems in similar ways with solutions that have already proven their value.

Network architecture is composed of nodes which are connected by a wire or radio link. This architecture has properties such as bandwidth and low level protocols for communicationg on the hardware.

Hardware architecture is in many ways like software architecture. Different respnosibilities are handled by optimised hardware. In a PC the hardware is split into RAM, CPU, harddisk and so on and these are all connected by a bus. The hardware is built in modules so that if the cusomer wants to upgrade the graphics card this can be easily done as long as the new graphics card complies with the interfaces of the motherboard. Externally visible properties are in this case bandwidth, clockspeed, number of pins in each slot and so on.

The architect decides to create a full architecture description before embarking on any implementation of the system. Discuss pros and cons of taking that approach.

Pros

- Makes the architect think about the whole solution in detail before taking the architecture in any direction.
- Makes it possible to discuss the proposed architecture with others architects and other stakeholders without wasting time implementing code that might change at a later time.
- All stakeholders will have well defined roles. Developers will know exactly how to implement the system.
- Easier to achieve the right balance between different qualitites.

Cons

- Difficult to understand any non-trivial system well enough to lay out the full architectural description in detail.
- Requirements change as time passes, hence a well planned system might be obsolete by the time it's implemented.
- Non functional requirements such as performance are often hard to estimate without running some prototype code imitating the full system, on the hardware the system will run at in the end.

1.2 Architectural description

Which structures does a system such as the above exhibit? Give examples of elements and relations pertaining to each structure.

Figure 1, 2, 3 and 4 show 3 different structures in 3 different views of the system. In Figure 1 and 2 elements in the structure are the packages and the dependencies between them are relations.

In the component and connector view in Figure 3 the components and the connectors are elements while the protocols that the elements use to communicate are relations.

In the allocation structure in Figure 4 the software and hardware units are elements and the software has allocated-to relations to the hardware it runs on. Furthermore the depencencies between the software elements are also relations.

Which viewpoints are relevant when describing the example? Give examples of partials views for each viewpoint.

Based on [Christensen et al, 2004] I have chosen to use 3 different viewpoints:

Module viewpoint which is a compiletime view showing how functionality is splitted into packages or development units.

Component and connector viewpoint which is a runtime view, which shows how functionality maps to components of the system and how they interact.

Allocation viewpoint which is a deploy time viewpoint that shows how software components in the system map to hardware units.

Argue for benefits and liabilities of describing software architecture via a box-and-line drawing such as the above.

Box and line diagrams can be a quick-and-dirty way of describing a system, and can be especially useful in brainstorms where the system hasn't started to take shape yet. It is also simpler to communicate with persons who aren't familiar with UML with box-and-line diagrams. Additionally this kind of diagram can serve as a summary of the whole architecture, even mixing some of the viewpoints.

But since there is no defined meaning to each of the boxes and the lines connecting them, box-and-line diagrams tend to quickly become a mess of different things, without the participants being aware of the "real" meaning behind the boxes. This can lead to confusions and people that talk past eachothers ears. Therefore it's preferred to use UML at least if the diagrams are kept for later reviews.

Discuss what architectural description would be needed if the system was to be realized as a service oriented architecture.

1.3 Quality attributes

Describe techniques for architectural requirements capture that are applicable to the above case.

Many of the requirements for a system that affect the architecture are non-functional and many of the most used methods for requirements capture (e.g. use-cases) result mainly in functional requirements.

[Bass et al., 2007] introduce quality attribute scenarios as a technique for capturing non-functional requirements, and even a common vocabulary for comparing different groups of quality attribute scenarios. The common vocabulary along with precise scenarios enables prioritization among often non-orthogonal requirements.

Give feasible architectural requirements for availability and performance for the Beer Web Store using such techniques (at least one for each quality attribute).

Performance

Scenario	Customers add beer to their shopping basket ten times in one second under normal operation.
Relevant Quality Attributes	Performance
Source	Users
Stimulus	Add to basket
Artifact	System
Environment	Normal operation
Response	Items added to the basket
Response measure	At least 10 pr. second.
Questions	
Issues	

Scenario	A customer checks out. His request is processed and the credit card validated with a latency of less than 10 seconds.
Relevant Quality Attributes	Performance
Source	Users
Stimulus	Check out
Artifact	System
Environment	Normal operation
Response	Check out processed successfully
Response measure	Less than 10 seconds
Questions	
Issues	

Availability

Scenario	The credit card validation service fails during normal operation. The failure is logged, and the user notified of the failure. The system continues to operate with no downtime.
Relevant Quality Attributes	Availability
Source	External
Stimulus	Payment fails unexpectedly
Artifact	System
Environment	Normal operation
Response	Abort the payment. Log the error. Continue in normal mode.

Response measure	No downtime.
Questions	
Issues	

Argue how tactics and/or styles may be used to resolve the requirements.

Tactics have been made for each type of quailty attribute scenario. That makes it simple and easy to start with a quailty attribute scenario and apply a tactic accordingly.

Discuss where service oriented architecture can play a role with respect to the Beer Web Store. Consider the quality implications.

The Beer Web Store can benefit from a service oriented architecture e.g. via using a credit card payment service that validates that the supplied information about the credit card, and then transfers the money from the customer to the shop. This way the shop itself don't have to solve the difficult but very common problem of credit card payment on the internet. The companies that provide such a service arguably have much better chance of implementing the solution in a robust, secure and performant way, than most small companies that have specialized in other aspects of software development.

It's also a quite plausible design to let parts of the database be replaced with services provided by the producers of the beer, so that the web site polls the producer instead of reading from the database. That way the producers could provide up to date information, detailed descriptions, add new labels and so on.

Reflect upon what the role of quality attributes in software architecture is.

Quality attributes define or indicate the non-functional requirements of the system.

1.4 Architectural design

Give examples of performance tactics respectively availability tactics that may be applied to the scenario. What are the consequences of applying these tactics?

Performance tactics

In this situation I would use the resource management tactic and introduce concurrency so that multiple payments can be processed at the same time. This also opens up for the possibility to scale up at a later time as the load on the web server increases with an increasing number of concurrent customers.

Availability tacitics

The quality attribute scenario states that the system should detect the fault, and then continue in degraded mode. So I would apply a fault detection tactic with exception handling.

The software architect decides to use a layered architectural style for the system. Discuss quality implications of this choice.

A layered design is a way of achieving information hiding at a greater level than with objects and components. This is especially true in a strictly layered system where each layer only communicates with the layer just "below" and just "above". A layered architecture often indicates that the different layers might run on separate machines.

By hiding details of the system we can achieve greater modifiability. Modifiability can be key to adding extra databases or webservers at a later point which can improve both performance and availability. On the other hand though, it is generally the case that indirection, which is a consequence of information hiding, hurts performance. Direct access to needed resources is, if seen in isolation, more performant than if the access is done through multiple layers. ok

Outline major elements of a possible component and connector structure of the Beer Web Store system.

1.5 Product Lines

How will the choice of a product line approach affect the architecture of the beer web store?

Discuss the benefits and liabilities of using a product line approach in relation to the case.

A product line approach will standardize architectural elements that different applications or parts of applications have in common. In other words the common elements will have to be made in a general and generic way, which requires the architect to consider the systems as a whole, to lay out where the different systems can benefit from the product line approach.

Both this analysis and the implementation of the more general system will initially come at a greater cost in form of time and money. Studies show that after about 3 products have been created from a product line the project breaks even comercially compared to creating individual products every time. So the product line approach should only be chosen if the company expects to sell at least 3 products based on the product line.

Since the initial cost of product lines is high it is very important to have the top management of the company agree on the approach. Othervise they might soon beceome disappointed in the apparently slow progress.

Relate Service Oriented Architecture and Product Lines.

?

Godhendt, men en del mangler

References

[Alexander et al. 1977] Alexander, C., Ishikawa, S., Silverstein, M. (1977). *A Pattern language*. Oxford University Press.

[Bass et al. 2007] Bass, L., Clements, P., and Kazman, R. (2007). *Software Architecture in Practice*, Addison-Wesley, second edition.

[Christensen et al., 2004] Christensen, H., Corry, A., and Hansen, K. (2004). An approach to software architecture description using UML. Technical report, Computer Science Department, University of Aarhus.

[Fielding 2000] Fielding, R. T., (2000) *Architectural Styles and the Design of Network-based Software Architectures, ch 1.* http://www.ics.uci.edu/~fielding/pubs/dissertation/software_arch.htm

[Perry and Wolf, 1992] Perry, D., Wolf, A., (1992) Foundations for the Study of Software Architecture. SIGSOFT softw. Eng. Notes, 17(4):40-52.