Practical Software Architecting

Tobias Glocker
University of Vaasa
Tobias.Glocker@uwasa.fi
Student Number: p87915

May 20, 2011

1 MANAGING ARCHITECTURE KNOWLEDGE

An architecture of a system contains "fundamental concepts or properties of a system in its environment embodied in its element, their relationships and in the principles of its design and evolution (ISO 42010)". There is no unique solution for a software architecture and thus the architectures are "inherently good or bad". For building the architecture of a system it is necessary to create, capture, share, distribute and understand the knowledge of an organization. "Architecting is a knowledge-intensive activity" that "produces and consumes knowledge artifacts" and "follows a reasoning process". (Avgeriou & Van Heesch 2011.)

Avgeriou & Van Heesch (2011) introduce the workflow and the concept of a backlog investigated by Hofmeister, Kruchten, Nord, Obbink, Ran & America (2006). The model of Hofmeister et al. (2006) is based on three main activities called architectural analysis, architectural synthesis and architectural evaluation. These activities "do not proceed sequentially" and thus the architects need to move from one activity to another, extending "the architecture progressively over time". For an architect it is impossible to "analyze, resolve, find solutions and evaluate the architecture for all architectural concerns simultaneously". No human mind is able to define all the goals, constraints, etc. at the beginning of the architecture phase. Most of the time the goals and constraints are better understood when the architecture grows. To manage such a "haphazard process" it is better for the architects to focus on "a

backlog of smaller needs, issues, problems they need to tackle, as well as ideas they want to use". The backlog helps the architect to determine the next step that needs to be done. For small projects a backlog can be a "list in the architect's notebook". In bigger projects a backlog "might be an electronic, shared spreadsheet". To get full benefits a backlog should be regularly updated with architectural concerns, negative issues or problems occurred in the evaluation process of the architecture and with ideas that came up. A backlog item can be considered as a decision statement. For each item an architectural synthesis is done. After this step the architects need to think of how the design decisions made for this item could be integrated with an existing set of design decisions. Backlog items "drive the architectural analysis and architectural evaluation". If a backlog item is "resolved in any way" (idea explored, requirement satisfied, etc.) it will be removed from the backlog and the architect will proceed with the next important item. An item is returned to the backlog when a problem is encountered. Hence, a "backlog is constantly changing". The main task of a backlog is to guide the architectural design flow through the three previous mentioned activities (architectural analysis, architectural synthesis and architectural evaluation). (Hofmeister, Kruchten, Nord, Obbink, Ran & Pierre America 2006.)

2 ARCHITECTURE PATTERNS AND SYSTEM QUALITY ATTRIBUTES

2.1 Software Architecture

Software design can be a very challenging and demanding process. To save time and development costs a very good software architecture must be created. Harrison (2011) introduces three important features that should be taken into account when designing software. One of these features is utility. Another feature is beauty, meaning that the designed software "must be pleasant to use". In addition the software should be reliable and should have an "adequate performance". Furthermore it is desired that the software provides a platform for "future evolution". Reliability, performance and providing a long-term platform belong to the third feature called firmness. Quality attributes play an essential role in the software architecture. The most important quality attributes are reliability, performance, capacity, usability, extensibility, portability and security. Since the architecture "is designed early in the project" it is essential to fulfill the quality attributes for the software that needs to be developed. Changing the attributes at a later state increases the development

time and costs. To avoid serious problems it is necessary to do early evaluations and analyses. In this process prototypes should be developed and if possible simulations should be done. It is also recommended to review architectural evaluations from others to learn from their problems and solutions. (Harrison 2011.)

2.2 Architecture Patterns

The most common architecture patterns are pipes and filters, layers, layers with shared services, micro-kernel, client server, broker, shared repository, blackboard, model view controller and presentation abstraction control. Pipes and filters process data sequentially. This pattern is mostly used for compilers, streaming data and assembly line support. In addition this pattern is quite flexible since it can have various configurations. It is very common to use layers due to its good portability and extensibility. In this pattern multiple abstraction levels are created. "Each layer communicates only with adjacent layers". The micro-kernel architecture pattern provides "a common plug-and-play interface for (usually) low level operations" and it has a good portability. Enhancements can be done easily but sometimes the performance can suffer. In the client server design pattern "a central server provides services to multiple clients". It is especially applied "among distributed applications". Moreover, "a variable number of clients" can be served and these "systems scale well by adding more servers". To ensure availability multiple servers can be used. The main difference between the client server and a broker pattern is "generally an extension to client server" pattern that "intercepts messages and does some preprocessing". In addition it provides also security features like a firewall or an authenticator. A shared repository is "a common component for keeping data". It is "accessed by multiple other components". Nowadays most of the databases are shared repositories. This architecture pattern considers performance, access and update issues as well as reliability and availability. The blackboard pattern is "a special purpose shared repository" with the capability of processing data. This pattern has the advantage that independent programs cooperate with each other by using a common data structure. The Model View Controller (MVC) architecture is used to manage user interfaces with three components. One of these components is the model that takes care of the main processing. It processes the commands received by the controller (handles user actions) and informs the view whenever a screen update is required. The presentation abstraction control architecture pattern is an "alternative to MVC".

It "handles different types of data presentations, usually simultaneously" and it contains "modules with different abstractions and representations of data". (Harrison 2011.)

2.3 Patterns as they appear in architectures

The architecture patterns discussed in section 2.2 play an essential role when a software architecture for a certain system is developed. It is necessary to find all the required architecture patterns. **Figure 1** shows an architecture of an airline booking system that consists of clients, a Wide Area Network (WAN), an application server and a database server.

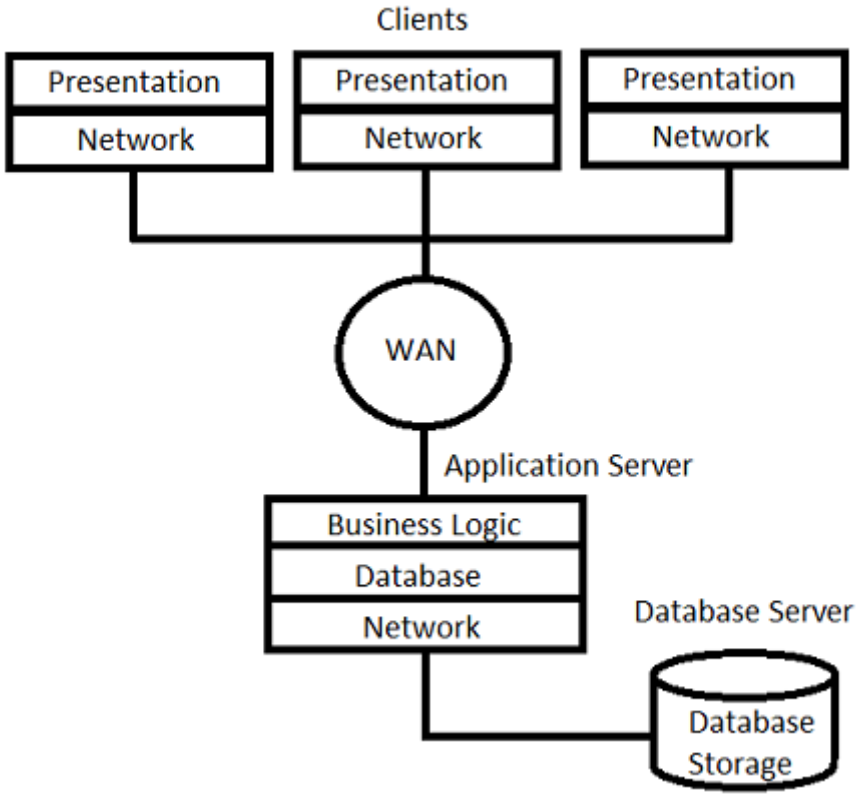

Figure 1. Airline Booking System.

It can be seen that this system contains at least two architecture patterns. Since there are clients and servers in the system, a client and server architecture pattern is required. Furthermore, the clients and the application server use a layer pattern architecture. The database storage could be a shared repository. To increase the

security of the system, authorization and authentication needs to be added. This can be done by adding a security layer on top of the business logic. Moreover, it is more save to store the names, passwords etc. in the database. (Harrison 2011.)

3 INTEROPERABILITY CHALLENGE IN DISTRIBUTED ARCHITECTURES

Complex IT products usually consist of "several system blocks", "several processor subsystems with several operating systems" and "several component vendors with their own software teams". Especially globally distributed systems like mobile services, corporate applications, remote access to smart thin-client products, cloud services and location based services belong to the category of complex IT products. An architecture landscape for globally distributed systems can be divided into four parts (see **Table 1**).

Table 1. Architecture landscape. (Suoranta 2011.)

Drivers	**Trends**
- Business ecosystem - Partnerships - Organizations - Product life cycle - Technologies	- From software to systems - Increasing distributed intelligence - Importance of open innovation
Environment	**Concerns**
- Smart spaces - Networked R&D - Connected products: - Global & Local	- Standardization vs. open innovation - Interoperability - Component models

There are three important features that need to be considered when developing globally distributed systems. One of them is modularity. Modularity means that a system is divided into a set of functional units also known as modules that can be composed into a larger application. The remaining two implemented features are functional abstraction and communication abstraction. Functional abstractions refer to a structured program where the flow of control should be kept as simple as possible. Furthermore the program should be constructed in such a way that it embodies top-down design in which a problem is decomposed into smaller problems.

Figure 2 illustrates the modularity and abstraction layers of a typical globally distributed system.

Figure 2. Modularity and Abstraction Layers of a Globally Distributed System.

Some embedded devices contain a service-oriented architecture (SOA). A SOA provides a flexible set of design principles. An embedded device that uses this architecture is divided into several subsystems that are connected with each other. This architecture allows the use of "heterogenous technologies" in a single device. Furthermore the device is "stepwise verifiable and testable" since every subsystem can be tested separately.

In the future complex IT products consist of a globally distributed system architecture with heterogenous platforms. It is possible that several components belonging to this system architecture are produced by different manufacturers. This leads to the disadvantage that many challenges must be conquered. To these challenges belong interoperability, new R&D models and new design paradigms. (Suoranta 2011.)

4 BIBLIOGRAPHY

Avgeriou, Paris & Uwe van Heesch (2011). Managing Architecture knowledge. University of Groningen. Lecture slides.

Harrison, Neil (2011). ARCHITECTURE PATTERNS AND SYSTEM QUALITY ATTRIBUTES. Utah Valley University and the University of Groningen. Lecture slides.

Hofmeister, Christine, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran & Pierre America (2006). A general model of software architecture design derived from five industrial approaches. Elsevier Inc.

Suoranta, Risto (2011). Interoperablitiy Challenge in Distributed Architectures. Notava Oy. Lecture slides.