

Towards Scalable and Adaptable Software Architectures

Mohamed E. Fayad
Dept. Computer Engineering
San José State University
San José, CA 95192-0180, USA
m.fayad@sjsu.edu

Haitham S. Hamza
Dept. of Computer Science & Eng.
University of Nebraska-Lincoln
Lincoln, NE 68888-0115
hhamza@cse.unl.edu

Huascar A. Sanchez
Dept. Computer Engineering
San José State University
San José, CA 95192-0180, USA
hsanchez@email.sjsu.edu

Abstract

Developing scalable and adaptable architectures that can accommodate evolving changes is crucial for reducing software development cost. To achieve scalability and adaptability, developers should be able to identify where and how new (current) layers will be added (removed) from the architecture. Failing to do so may lead to software architectures that require a considerable modifications when the system evolves or changes due to new or added requirements. In this paper, we address the problem of developing scalable software architectures that can accommodate new and/or modified requirements without the need for re-developing the architecture from scratch. The approach is demonstrated through a case study.

1. Introduction

When businesses experience substantial increments in their services' demands, their main concern is their application architecture's ability to scale to cope with these loads. The architecture is required to efficiently scale, and adapt in such a manner that it will fit in both constrained and unconstrained environments; yet still being able to take full advantage of the available resources.

Scalability as a term, sometimes, is used to define the capacity of employees to handle an increased amount of tasks in their workplace [9]. From the perspective of software, *scalability* can be defined as the capacity of software systems to handle increasing loads, or demands, by users at certain time [9].

Scalability is concerned with the ability of the architectures to scale Up and Down in order to adapt to evolving requirements. Scaling Up and Down can also be referred as “Upward and Downward Scalability”. Upward scalability is

defined as the capacity of handling increasing demands by adding new layers or functionalities, while downward scalability is defined as the ability of the architecture to adapt itself to a more constrained environment, by disabling certain functionalities. Though the software architecture is being used in a more compacted environment, it is still able to take full advantage of the available resources.

The idea behind the achievement of Vertical Scalability in software architectures is the partitioning of the entire architecture functionality into individual components that can be used separately. However, this process is still not addressed and it is difficult to implement. Figure 1 illustrates the two sides of Vertical Scalability.

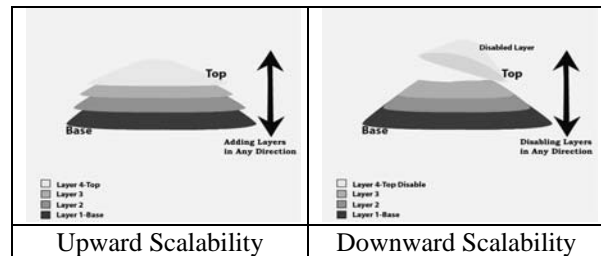


Fig. 1. Vertical scalability-two perspectives

Horizontal Scalability, on the other hand, emphasizes the ability of the architecture to extend their boundaries by establishing connections with other software architectures in an efficient manner. The architecture attached to a primary architecture may be considered “The extended leaves of the primary architecture”. Horizontal Scalability can be achieved in two directions, which we refer to as: Scaling Out and scaling In. These directions are referred as “Extensibility and Reduction”. Extensibility is the capacity of software architectures to glue external architectures (leaves) to their core structure, creating a synergy between

these dissimilar architectures. Reduction, on the other hand, is the action of ungluing those “leaves” from a particular architecture without provoking a reciprocal harmful impact on their internal structure. Figure 2 illustrates the Horizontal Scalability.

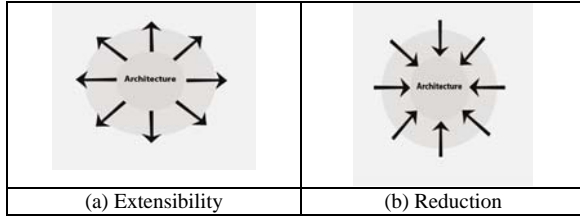


Fig. 2. Horizontal scalability-two perspectives

Scalability has been addressed by using conventional approaches, but with some limitations. For instance, new trends on the implementation of Scalability in software architectures are emphasized merely on the achievement of *Upward Scalability*, dismissing completely *Downward Scalability* and *Horizontal Scalability*. They rely on three elements: Increasing Speed and Capacity, Improving efficiency, and shifting or reducing the loads [8].

Based upon these elements, designers/ developers need to pinpoint relevant information along with the right techniques to come up with a proper scalable architecture design. Usually, the main techniques applied to reach the former elements are: Use of Faster Machines, Create a Cluster of Machines, Use Appliance Servers, Segment the Workload, Batch Requests, Aggregate User Data, Manage Connections, Cache Data and Request [8, 10]. These Techniques address the three core elements to be considered in scalability (Upward) in this order: For Increasing Capacity and Speed: Use of Faster Machines, Create a machine cluster, and Use of a Appliance Servers; Improve Efficiency: Use of Appliance Servers, Segment the Workload, Batch Request, Aggregate user data, Manage Connections, Cache data and request; and for Shifting and Reducing Cost: Segment the Workload, Cache data and requests [8].

Even though, these techniques try to guarantee scalability, they tend to offer limited scalability due to: (1) They are solely oriented to reach one side of Vertical Scalability (Upward). (2) Their validity, usually, relies on special hardware. These observations have led us to confirm that implementing full scalability is not an obvious task in software development, and especially with conventional approaches [6].

In this paper, we address the problem of developing scalable software architectures that can accommodate new and/or modified requirements without the need for re-developing the architecture from scratch.

The paper is organized as follows. Traditional design approaches and the stability design approach are discussed in Sections 2 and 3, respectively. An illustrative case study is given in Section 4. Conclusions are presented in Section 5.

2. Existing Design Approaches

A major obstacle in developing scalable and adaptable architectures with existing design approaches is in their resounding lack of robust, systematic procedures for effective architecture modularization. For instance, when dealing with Vertical Scalability, it is not clear which elements or layers have to be kept or dropped to reach an efficient result [7]. In Horizontal Scalability, existing approaches don’t facilitate the detection of the architecture’s binding points, where it binds to/unbinds from other external architectures. These factors will incapacitate the ideal architecture not only on its deployment, but also in its capacity to adapt to multiple environments (Constrained and Unconstrained). Additionally, these factors will turn the resulting architecture into an excellent host for “fast spreading” contamination problems (“Ripple Effect”) at the time of scaling it. The spreading ripple effect will be shown in figure 3.

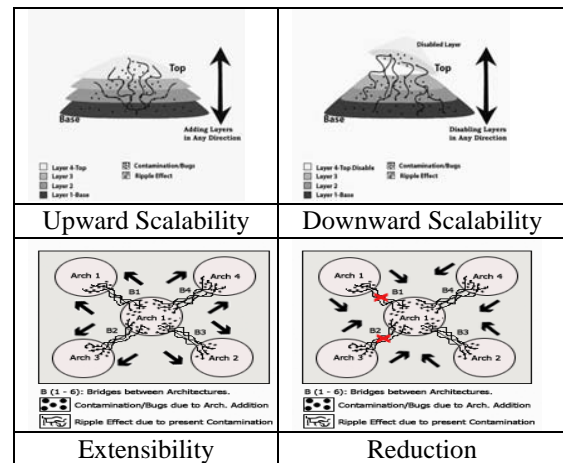


Fig. 3. Spreading Ripple Effects

Existing difficulties experienced when using traditional approaches are listed below using a

detailed “Cause and Effect” table. The effects of using conventional approaches over the implementation and deployment of Timeless and Scalable Architectures would be illustrated. Table 1 shows these causes and effects of Traditional approaches on spawning Timeless and Scalable architectures.

As shown in Table 1, the effects’ section is partitioned into two categories; the first category describes problems experienced by Analysts/Designers/Developers at the moment of implementing scalable Architectures. The second category illustrates the impact in the architecture itself. These consequences can be seen in a simplified way using Figures 3.

Scalability	Cause		Effect	
	Guidelines	Processes	Implementation	Architecture
	Availability	Usage	Impact & Complexity	Occurrence
Up & Down	1,2,3,4 - None	1,2- High	1,2,3 - High Impact 4- Med. Effort/High Impact 5- High Effort & Impact	1,2,3- High
In & Out	1,2,5- None	1,2- High	1,2,3- High Impact	1,2,3 - High
Problem Definition	1.No Methodology 2.Unclear Direction 3.No Layer Identification 4.No Connection Points 5.Layer Boundary Shortage	1.Non-Systematic Process 2.Ad-Hoc	1.Impossible test scenarios. 2. Low Cohesion/Unstable Structure 3.No Cost/Time Effective 4. Scaling Up 5. Scaling Down	1.Architecture Collapse 2.Embedded Contamination 3.Ripple Effects

Table 1. Limitations with conventional design approaches

Those illustrations show the ripple effects experienced by the developed architecture using Traditional approaches. Constant Developer involvement will be required to safeguard the architecture from collapsing due to these ripple effects. Unfortunately, traditional approaches do not provide any help to latter case. They do not offer the right tools or processes to accomplish Scalability when addressing large changes in problem size [7]. Additionally, due to their lack of flexibility, traditional approaches are unable to handle the diversity of methods employed when achieving scalability in different problem domains [7].

3. Software Stability Model Approach

Software Stability Model (SSM) [3] classifies the system architecture into three layers (Figure 4): EBTs, BOs, and IOs. Enduring Business Themes (EBTs) define the elements of the system that remain stable internally and externally [3, 5].

Hence, they are conceptual classes which represent the core aspects of the problem domain that are not affected by the forces of change over time. One good example would be “having *scalability* in our system as core aspect or endure business theme (EBT).” On the other hand, Business Objects (BOs) are objects that are internally stable but externally adaptable via Hooks [3, 5]. They are semi-tangible classes that provide the essential means for carrying out EBTs – they are the work horses of any system from a domain-neutral perspective [3, 5]. The last artifact of the Stability Models approach is “The Industrial Objects (IOs).” Industrial Objects are the context-specific classes of the software system represented in the model [3, 5]. They are tangible classes that can change internally and externally over time. Concretely speaking, they are the context specific functionality attached to BOs to handle particular needs in a determined context.

SSM provides a stable, reusable, and scalable core for the scalable architectures. This core is represented by EBTs and BOs. The idea of using EBTs and BOs for the development scalable architecture relies on the stability properties of both [3]. EBTs and BOs can form the basis for an efficient modularization of the entire architecture, allowing a portable, adaptable, and customizable behavior in the architecture.

To evolve specific application from the stable core (i.e. EBTs and BOs), IOs can be attached to the core by means of extension points, or Hools, defined in the BOs [2].

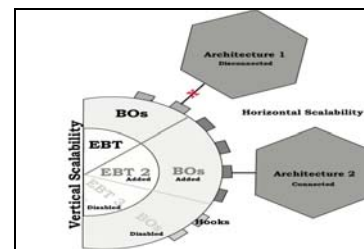


Fig. 4. SSM approach

The basis for stable software architectures is not driven by the generalization-specialization problem, in spite of that there are some counted cases in SSM where BOs seemed to be extended into IOs. However, it is also shown that there are many cases where IOs are associated with BOs by means of simple associations and aggregations relationships. Regardless the association type with BOs, IOs would be utilized, by aggregating them into BOs’ hooks, to enable, disable, extend, change, add, and

remove behavior of the targeted BO (the hook's owner) [2, 11].

4. Case Study

We developed a simple e-commerce architecture using both traditional development approach and the stability approach. The scalability of the two developed models will be then compared under different possible evolving changes.

4.1 E-Commerce Architecture with Traditional Approaches

Figure 5 shows the traditional model for simple e-commerce architecture. In this architecture, the Customer navigates through the Catalog of products that are retrieved from a Database. Then, the Customer selects a product to be purchased, which is also retrieved from the Database. This product is automatically placed in the ShoppingCart System. The customer decides to proceed with the purchases by checking-out the Order. The Customer needs to provide first his/her Credit Card information before completing the Order.

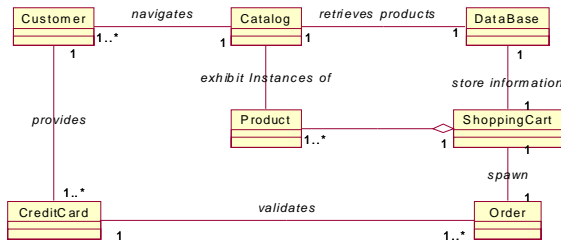


Fig. 5 Simple E-commerce Architecture

Now, consider the scenario where we want to change the payment method in this architecture. These new conditions will require major changes in this conventional model due to a common tendency of Conventional Models to be remodeled every time a change is incurred [6]. In this case, new artifacts need to be included in our traditional model to handle heterogeneous payment methods (i.e. Cyber Cash, E-Checks, etc).

The occurrence of changes is proportional to the endless increment of system demands. Therefore, a constant adjustment of our conventional e-commerce architecture's model and its artifact services would be required every time new changes arise. These adjustments may affect the consistency of the entire model: lost of simplicity and readability, it will turn unstable, more complex, and difficult to follow (Figure 6). As a result, any

improperly modification in one class' services may have an effect on other classes' services connected with.

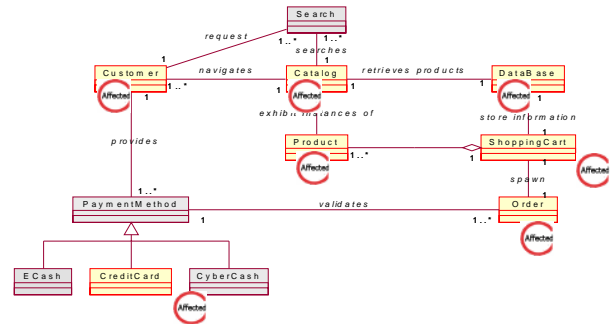


Fig. 6. Artifacts to be modified due to Requirements Changes

4.2 E-Commerce Architecture with SSM

Now, we develop the same e-commerce architecture, using Software Stability Concepts. To do so, firstly, we need to identify the purpose, and goals of this system (EBTs). Then, we identify the core abstractions of the system's processes (BOs). And finally, we need to identify the physical entities of our model (IOs).

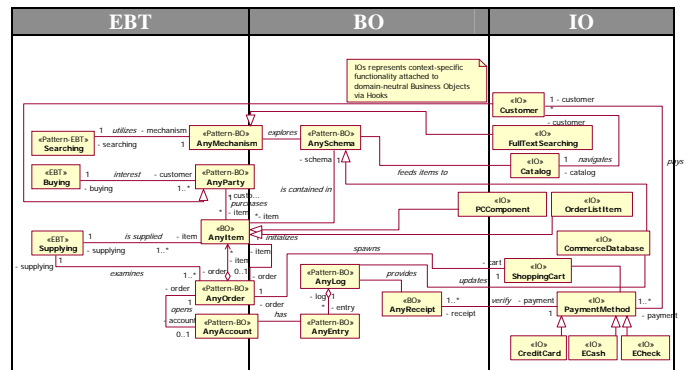


Fig. 7. Stable Model of our E-Commerce Architecture

In the developed stable model, scalability can be accomplished in the four scaling directions: upward, by adding more independent functionalities into the system's architecture (by adding one or more enduring concepts EBTs, their respective BOs, and IOs into the architecture model), downward, by disabling some functionality from the system's architecture model (By removing an EBT and its respective BOs and IOs) with the purpose of fitting in a more constrained environment, extensibility, by connecting heterogeneous architectures or "extended leaves" to particular architecture through the utilization of Hooks [2, 11], to augment its

capacity of sharing resources with other architectures, and reduction, by disconnecting those leaves from a particular architecture without provoking a harmful impact on the core structure of each one of the involved architectures.

It is noteworthy that any single representation of EBTs, their associated BOs and IOs, represents an independent functionality of the group of functionalities in the entire system's architecture. Each functionality, by their definition, is self-controlled. Nevertheless, it collaborates with other functionalities in order to accomplish one or more tasks in the architecture. To understand how these independent layers or functionalities are laid out, we will use the SSM of the E-commerce architecture shown (Figure 7). For simplicity, we use only two independent functionalities to demonstrate that particularity of SSM (Figures 8 and 9).

4.3 Scalability Test Scenarios

Most likely, changes that cause software architecture to fail are product of the natural evolution of its businesses processes and the modification of its underlying requirements [6]. In this Section, we assume a changing in the requirements of the e-commerce system and test how both the traditional and the stable architecture can be modified to accommodate those changes.

Example 1: Provides Customer Services or Assistance in the architecture, Content Management and Product Bidding Capacity.

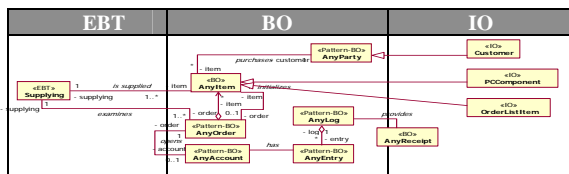


Fig. 8. Supplying/Order Handling Functionality in the SSM

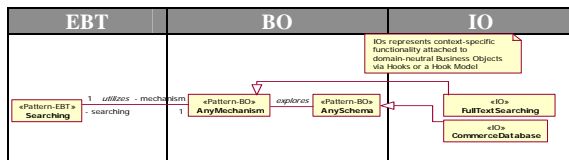


Fig. 9. Searching Functionality of our Stable Model

In order to either keep current customers or attract new ones we need a customer service capability. Also, we need to enhance the overall architecture by adding Content management and product bidding applications. Figure 10 shows how SSM adapts to new requirements.

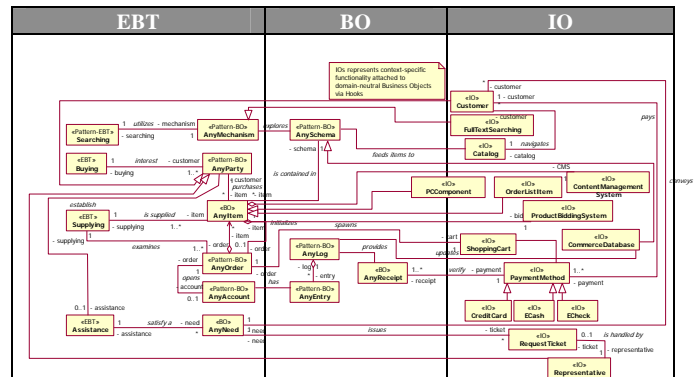


Fig. 10. Adding new functionalities to the stable model.

Example 2: Disabling Item Supplying/Order Handling functionality and Content Management Application.

Imagine that another company acquired this e-commerce application to substitute its old one. This company already has a merchant account and doesn't want to disable it, and has showed not interest in the content management application. To solve this we need to disable ItemSupplying functionality and content management from the architecture. Figure 11 shows this adaptation.

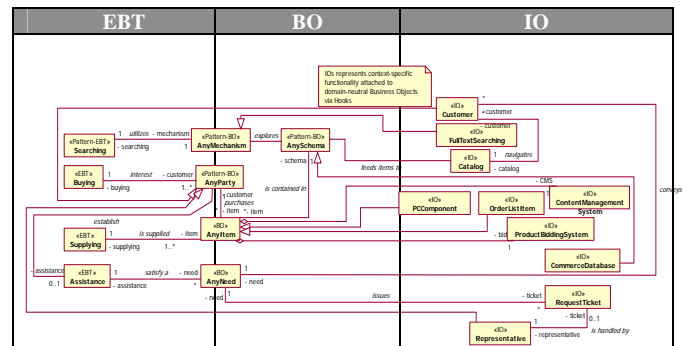


Fig. 11. Disabling some functionalities in the stable model.

4.4 Discussion

The achievement of a scalable E-commerce through the utilization of Traditional approaches rest upon a fundamentally different set of ad-hoc (Non-systematic) processes than the ones used in SSM: e.g. a greater focus on adding or removing classes into/from the system architecture to cope with new requirements; creating high dependency among all the classes involved. At first, this action seems to be very inoffensive, but remember, requirements are not fixed; they are generally

volatile. Therefore, changes (adding-removing) in determined classes in the architecture will open doors for serious ripple effects in the architecture (i.e. the addition of classes such as Search, and PaymentMethod). In contrast, when using SSM to generate a scalable E-commerce architecture, we rely on a set of systematic processes oriented to welcome changes in the architecture over time: e.g. a greater focus on adding or removing self-independent functionality (concerns) into/from the architecture to cope with arisen requirements, without creating high dependency among them (i.e. adding/removing Item Supplying/Order Handling Functionality). The key factor in SSM is that each functionality represents *an independent* component that can be added/removed in/from the architecture with ease and with no ripple effects.

Traditional approaches have considered differently the adaptation of the Scalable e-commerce architecture to couple/decouple with/from external software systems than SSMs: e.g. adaptation relies on non systematic approaches in where there is not clear idea of the distinctive connection points in where the architecture can be glued to or unglued from, as a result the accuracy of which key classes to modify and how to modify them, would be drastically affected (i.e. noteworthy increment of Costs and Time consumed to cope with these issues). In Contrast, SSM addresses this adaptation of the scalable E-Commerce Architecture from a different perspective: e.g. Stability model explicitly divide the architecture into the endured and stable knowledge, which is conformed by EBTs and BOs and their mutual connections, and the changeable and tangible (unstable) knowledge (IOs), which are attached via hooks [2]. Hooks are the connection points where our stable and scalable E-Commerce Architecture can achieve Horizontal scalability in a cost-effective manner.

6. Conclusions

In this paper we demonstrated, through a simple case study, that developing architectures using

traditional approach may not allow the desired flexibility to accommodate future requirements. As requirements evolved, the simplicity and readability of the traditional architecture were affected, leading to more complex architectures. On the other hand, software stability seemed to provide an approach for incorporating scalability features into developed architectures. Thus, architectures can scale up or down, and scale out or in to accommodate evolving requirements.

7. References

- [1] Froehlich, G., Hoover, H.J., Liu, L. and Sorenson, P.G. "Reusing application frameworks through hooks", in Object-Oriented Application Frameworks, M. Fayad, D. C. Schmidt, and R. Johnson eds., John Wiley, 1998, in press.
- [2] M.E. Fayad, R. E. Johnson (Editor), D.C. Schmidt (Editor), "Building Application Frameworks: Object-Oriented Foundations of Framework Design", Publisher: Wiley, John & Sons, Incorporated, September 1999.
- [3] M.E. Fayad and A. Atman "An Introduction to Software Stability." *Comm. of the ACM*, Vol. 44, No. 9, Sept. 2001.
- [4] M.E. Fayad and S. W. " Merging Multiple Conventional Models into One Stable Model", *Comm. of the ACM*, Vol. 45, No. 9, Sept. 2002
- [5] M. Laitinen, M.E. Fayad, Robert P. Ward. "The Problem with Scalability", *Comm. of the ACM*, Vol. 43, No. 9, September 2002.
- [6] P. Lee. "R/KYV V9 - Scalability." Valid Information Sys 02, www.valinf.com, August 2002.
- [7] Scalability's New Meaning, www.evolt.org.
- [8] R. Allen, R. Douence, and D. Garlan, "Specifying and Analyzing Dynamic Software Architectures", *Proc. Conf. on Fundamental Approaches to Software Engineering Portugal* (March 1998).
- [9] D.E. Perry, A.L. Wolf, "Foundations for the Study of Software Architectures", *ACM SIGSOFT, Software Engineering Notes* vol. 17 no 4, Oct 1992 page 40.
- [10] R.t Allen, D. Garlan, "A Formal Basis for Architectural Connections", Carnegie Mellon University.