Balanced MVC Architecture for High Efficiency Mobile Applications

Hyun Jung La and Soo Dong Kim[1]
[1]Department of Computer Science
Soongsil University 511 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
[e-mail: {hjla80, sdkim777}@gmail.com]
*Corresponding author: Soo Dong Kim

Received December 1, 2011; revised January 30, 2012; accepted March 6, 2012; published May 25, 2012

Abstract

Mobile devices such as Android devices are emerging as a convenient client computing device with mobility and context-sensing capability. However, the computing power and hardware resource of the devices are limited due to their small form-factor. Consequently, large-scaled applications could not be deployed on these devices. Nonetheless, if the large-scaled applications are deployed and executed on the devices, high performance of the applications cannot be guaranteed. To remedy the limitation in terms of performance, it is inevitable to let some heavy-weight functionality executed on the server side and let a client application invoke the functionality in the server. To realize this kind of mobile applications, we adopt well-defined architecture design principles; being thin-client, being layered with Model-View-Controller (MVC), and being balanced between client side and server side. By adopting the principles, we propose a unique, ideal and practical architecture for mobile applications, called *balanced MVC architecture*. By considering the principles, key design considerations of realizing balanced MVC architecture lie in functionality partitioning. Hence, we define key criteria of determining the degree of performance. And, we define a method to design a balanced MVC architecture which embodies functionality partitioning for high performance, and a simulation-based evaluation method of balanced MVC architectures.

Keywords: MVC Architecture, mobile application, efficiency, design method, evaluation method

this work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2009-0076392). We thank the anonymous reviewers for their constructive feedback which greatly helped us to improve the quality and presentation of this paper.

http://dx.doi.org/10.3837/tiis.2012.05.0010

1. Introduction

Mobile devices with iOS and Android provide software functionality as well as cell phone capability. With the advent of more powerful mobile devices, they have been emerged as a convenient client computing device not just for casual personal computing but also for enterprise computing. The potential usage of utilizing mobile devices goes beyond conventional personal computing due to the mobility, networking, and context-sensing capability.

However, mobile devices have a major drawback of limited resources mainly due to the small form-factor. Consequently, large-scaled applications consuming a large amount of resource could not be deployed on the devices. To overcome the limitation and to maximize benefits of the mobile devices, it is inevitable to run some computation-intensive functionality on a server side and let mobile clients invoke the functionality [1][2]. More specifically, there should be architectural guidelines for designing mobile applications with optimal distribution of functionality and dataset over the client side and the server side. This could potentially achieve higher QoS than standalone mobile applications.

Model-View-Controller (MVC) is a widely adopted layered architecture pattern for typical enterprise applications, and it provides a number of benefits. However, all three layers of MVC could not be allowed to mobile devices for applications with high functional complexity. Hence, in this paper, we propose an extended version of MVC for mobile applications, *Balanced MVC* architecture. The basic notion is to adopt MVC layers on both the client side and the server side, and to take a number of benefits while remedying the resource constraint problem of mobile devices. The term *balanced* in this paper is to partition the functionality and datasets between two sides by considering design criteria (given in section 4) in the process of designing mobile application architecture. It does not include the behavior of dynamical adjusting workloads among processors and data stores, which is a common service provided by middleware products such as EJB.

Within the balanced MVC, we identify its five patterns of assigning three MVC layers to the two sides. Each pattern yields different level of quality aspects, especially *performance*. Hence, we define key criteria for determining the *performance* as defined in ISO/IEC 9126 [3], and the criteria are used to derive the optimal architecture design for target application. In addition, we define metrics for evaluating mobile application architecture, not the application.

The paper is organized as the following. We first present the balanced MVC architecture and five more specific patterns of the architecture in section 3. Then, we define key criteria of determining the performance of mobile applications with balanced MVC in section 4, which can be used to designing and evaluating the architecture. And, we present a systematic method to apply the architecture in section 5, and a simulation-based evaluation method to assess whether the architecture design is fully satisfied with the criteria in section 6. Finally, we present three case studies; showing how the architecture is applied in practice, showing how correctly the architecture evaluated, and showing comparison results with other conventional patterns.

With the proposed architecture and design guidelines, we believe mobile applications with high complexity can be effectively designed, and yield high performance and low resource consumption.

2. Related Work

We summarize representative work on partitioning the application functionality, performance evaluation for mobile application architecture, and methods to design mobile applications.

For partitioning mobile application functionality, Tergujeff's work proposes service-oriented architecture (SOA) for lightweight mobile devices through a survey of enabling technologies, programming interfaces, and supporting devices [4]. Based on the survey, they present a demonstration architecture mainly based on JSR 172. Natchetoi's work presents a *lightweight* service-based architecture for business applications running on J2ME enabled devices [5]. They focus on devising design methods that cover important features of mobile devices; minimizing data transferred to and stored on the device, pro-active data loading, and security. Ennai's work presents an architecture of service-oriented framework that satisfies deploying lightweight service-based applications, adapting services and devices to user' contexts, and utilizing an intelligent invocation of services [6]. The architecture consists of clients, *mobileSOA* devices layer, virtualized services, and mobile device platform components. Each component is interoperated with each other to support dynamic service discovery, context-aware service portioning, and asynchronous service invocation. Kumar and his colleague address cost benefits of applying cloud computing to mobile devices [7]. They present a simple equation to calculate the amount of energy saved through computation *offloading*. The equation result implies that offloading is beneficial when large amount of computation is needed with a relatively small amount of communication. Their approach only considers offloading the whole functionalities to the server side. Hassn et al propose architecture framework for hosting Web services on a mobile device [8]. Based on the type of mobile web services, this work defines *Backend node based scheme*, *Intermediate node based scheme*, and *Forwarding node based scheme*. However, this work does not consider how to partition complex tasks are partitioned and how to integrate the result of partitioned tasks. There are several works on considering how to balance loads over the network such as Tran's work [9] and Mateo's work [10].

For performance evaluation for mobile application architectures, Ryan's work presents a set of metrics for measuring runtime efficiency of context aware mobile applications [11]. They first identify attributes that can affect runtime efficiency in terms of performance and resource utilization. Then, they establish hypotheses which show interrelationship between the attributes and perform experiments to demonstrate those interrelationships. And, they presented adaptation based methods to maintain a certain level of efficiency by using the metrics. Their metrics are used in managing context aware mobile applications, rather than in designing them. Aquilani's work presents a method to evaluate an expected performance of software architecture [12]. First, they present a method to define a performance evaluation model with architecture description. They utilize *Queuing Network Model* for defining the evaluation model. With the model, they evaluate performance of the software architecture by using metrics such as response time and throughput. They focus more on defining the performance model.

For design of mobile applications, Abrahamsson's proposes a methodology called *Mobile-D,* which is an agile approach to developing mobile applications to cope with technical constraints of the mobile environment [13]. The approach is based on development practices borrowed from *eXtreme Programming*, *Crystal methodologies*, and *RUP*. The nine phases and disciplines of *Mobile-D* are derived from conventional software development processes and relevant supporting activities. Rahimian's presents an approach to develop mobile software systems using *Hybrid Methodology Design* [14]. First, they identify unique requirements and

constraints associated with mobile systems. Based on these requirements, they define a nine-phase methodology. In addition, there are several works to propose a set of instructions for developing mobile applications by considering specific aspects. Häkkilä's work [15] focuses on proposing ten design guidelines for context-awareness aspect of mobile applications. Sá's work [16] presents design methods to gather context information, to develop prototype for evaluation, and to evaluate mobile applications. Ayob's work [17] presents a set of design guidelines to design user interfaces for mobile applications. Although these works present a design process covering a whole life-cycle, they do not consider designing mobile applications with important quality aspects such as high efficiency.

In summary, the works mainly focus on applying the service-based architecture to mobile applications and addressing the issues in designing the architecture with high-level instructional descriptions, which only covers overall architecture for the server side. And, evaluation methods tend to less focus on mobile application itself. Our work is to focus more specifically on designing and evaluating mobile applications with thin-client MVC.

3. Balanced MVC Architecture

As earlier discussed, to enable to execute functionalities of large-scaled applications with mobile applications, it is better to locate functionalities with higher complexity on the server system and let mobile application invoke the functionalities through the network. For this type of mobile applications, we define a new architecture, *Balanced MVC Architecture*, which is an extended MVC architecture [18][19] where client and server systems embody its own separate layers so that the degree of overall performance can be maximized with limited resources.

3.1 Key Elements and Their Relationships

In the mobile application architecture, client and server systems embody its own separate layers. **Fig. 1** shows key elements and their interrelationships of the mobile application architectures. There are two components in the client application. *C.View* is the layer providing user interface for mobile users, and *C.Control* carries out business process logics for the client application. All the functionality exposed to the users should be defined in *C.Control*, which invokes methods of classes in *C.Model*. *C.Model* contains entity-type classes which manage data for a specific mobile user and tend to be temporarily stored.

For the server system, there are two components. *S.Control* runs business process logics that invoke public methods of *S.Model* and can be reused by multiple mobile users. *S.Model* contains entity-type classes which manage persistent data for all the mobile users. Note that there is no view layer in the server system since the server system plays a role of providing functionality to the client system only interacting with the client system.

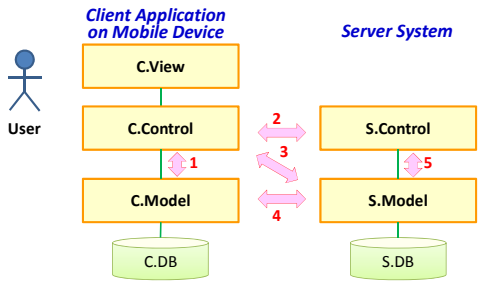


Fig. 1. Elements and interaction paths in mobile application architecture

There are five interaction paths in the architectures. Path 1 is an interaction between *C.Control* and *C.Model*, not requiring any communications with server system. This is applicable to functionality which is invoked on the client system without relying on the server system. Path 5 is much similar to Path 1 only except for the interaction parties.

Path 2 is an interaction between *C.Control* and *S.Control*. This is applicable to the case that the functionality of *C.Control* is fulfilled with the support of the *S.Control*. Path 3 is a direct interaction between *C.Control* and *S.Model*, without going through *S.Control*. This is applicable to the case that the *C.Control* needs to update the objects in *S.Model* efficiently so that network overhead is reduced. Path 4 is an interaction between *C.Model* and *S.Model*. This is useful to synchronize the states of two corresponding objects to maintain the state consistency.

3.2 Patterns of Balanced MVC Architecture

One of the challenging tasks in applying this mobile application architecture lies on the decision about where the functionality of business processes (i.e. control layer) should be allocated. We can consider three cases; business processes allocated to *C.Control*, allocated to *S.Control*, and balanced between *C.Control* and *S.Control*. And, another important decision is whether *C.Model* is needed. By considering all the concerns, we figure out the five architectural patterns for mobile applications, as shown in **Fig. 2**.

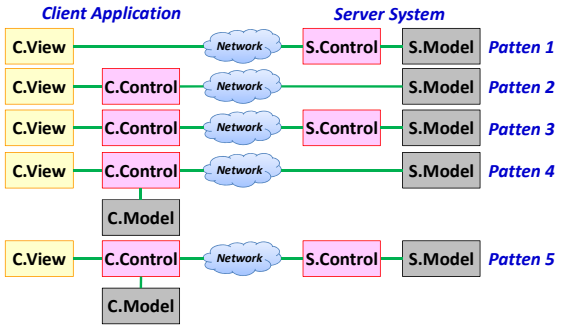


Fig. 2. Options for load balancing with balanced MVC architecture

In *Pattern 1*, there is only *S.Control* which implies all the business processes are executed on the provider side. This will fully satisfy the criterion of being thin on the client side. However, every interaction with users will have to go through the network.

In *Pattern 2*, there is only *C.Control* which implies all the business processes are run on the client side. Compared with *Pattern 1*, much larger amount of resource consumption is required in the client side. However, this pattern would be ideal for situations where the user interaction is intensive, and the interaction between *C.Control* and *S.Model* is relatively lower.

In *Pattern 3,* there are both *C.Control* and *S.Control*, which implies business processes are partitioned to both sides. If there are a large number of interactions between *C.Control* and *S.Model* in *Pattern 2*, this pattern would be well applicable because the number of interactions is minimized by putting *S.Control* which interacts with *S.Model*. We expect to have a high parallelism on *C.Control* and *S.Control*.

In *Pattern 4*, there are *C.Control* and *C.Model*. This configuration would be well applicable to situations where *C.Control* has intensive interaction with some parts of *S.Model*

so that the parts are replicated to *C.Model*, and it is expected to have a high parallelism between two model groups.

In *Pattern 5,* there are both *C.Control* and *S.Control*, and also *C.Model*. This configuration would be selected in situations where there exists a high coupling/dependency between each pair of *Control* and *Model*.

4. Key Criteria for Designing Balanced MVC Architecture

In this section, we derive key criteria that play important roles in designing architecture for mobile applications. We consider performance as a first-citizen quality attribute since limited resources of mobile devices severely affects overall performance of mobile applications. In this paper, we consider *performance* as *efficiency* defined in ISO/IEC 9126 [3], which is defined as the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. By adopting the definition of efficiency in ISO/IEC 9126, we consider two sub-quality aspects for evaluating performance, *time behavior* and *resource utilization*.

4.1 Time Behavior

Due to functionality partitioning of balanced MVC mobile applications, it is inevitable to incur additional costs such as network cost. Correspondingly, the costs result in decreasing overall performance. Hence, mobile applications with balanced MVC architecture should be efficiently designed to incur as fewer amounts of additional costs as possible. That is why we drive *time behavior* as the first criterion.

We now define three key factors which have high impacts on the overall value of time-related cost of balanced MVC mobile applications as shown in **Fig. 3**.

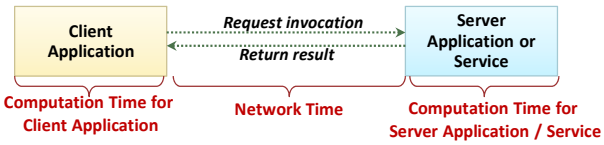


Fig. 3. Key Factors for *Time Behavior*

Factor 1. Computation Time (for Client Application and Server Application) : This is the cost spent in executing functionality on either client application or server system. To achieve high efficiency, it is natural to minimize a cost to compute functionality. That is, the performance is directly related to a complexity of the executed functionality which controls a set of data. Hence, the performance is affected by the following factors;

- The functionality complexity (*F*): A main cause to increase computation cost lies in the complexity of the functionality. If mobile applications execute extremely complex computation, they spent a lot of time because they have a low speed of CPU clock. More specifically, this factor depends on the number of input parameters manipulated (*Size(InParam(F)*) and the number of operations performing the functionality (*NumOp(F)*) [20][21].
- The size of data required by a mobile application (*Size(D)*)
- The complexity of managing the data: In addition to the size of data manipulated itself, the complexity of managing the data also affects the computation. This complexity is

determined by complexity of data query (*Complexity (QueryOp((D))*) and the number of access the database (*Freq(QueryOp(D)*)

Let *Cost_to_Compute(F, D)* be the computation cost of the functionality *F* to manipulate the required dataset *D*. This cost is evaluated with a following equation;

$$Cost\_to\_Compute\ (F,D) \propto (Cost\_to\_Compute\_with\_DB(F_i, D) + Cost\_to\_Compute\_without\_DB(F_j)) / (ClockSpeed)\ . \quad (1)$$

F is classified with two kinds of functionality; F_i working *with DB* and F_j working *without DB*. Hence, *Cost_to_Compute* is proportional to a sum of computation cost with accessing DB (i.e. *Cost_to_Compute_with_DB*), computation cost without accessing DB (i.e. *Cost_to_Compute_without_DB),* and inversely proportional to the *clock speed*. And, *Cost_to_Compute_with_DB* and *Cost_to_Compute_without_DB* are evaluated by considering the mentioned factors as following;

$$Cost\_to\_Compute\_without\_DB(Fj) \propto Size(InParam(Fj)) * NumOp\ (Fj)\ . \quad (2)$$

$$Cost\_to\_Compute\_with\_DB(Fi, D) \propto Size(InParam(Fj)) * NumOp\ (Fj) * Size(D) * Complexity(QueryOp(D)) * Freq(QueryOp(D))\ . \quad (3)$$

In the balanced MVC architecture, we need to consider functionality cost on client side and functionality cost on services.

- $Cost\_to\_Compute(F_{client,}\ D_{client})$ for client side
- $Cost\_to\_Compute(F_{server,}\ D_{server})$ for provider side

According to [7] and [22]'s explanation on offloading, we can verify potential benefits of balanced MVC mobile applications since clock speed of a server system is much faster than one of a mobile application.

$$Cost\_to\_Compute\ (F_{client}, D_{client}) + Cost\_to\_Compute(F_{server}, D_{server}) \leq Cost\_to\_Compute\ (F_{client}, D_{client}). \quad (4)$$

Factor 2. Network Time: This is the cost consumed in sending invocations and receiving results between client and server parts. When F_{client} invokes functionalities denoted by $F_{provider}$, a set of input parameter may be passed with the invocation and a result is returned over the network which has its own bandwidth. At this time, the network overhead is affected by the following factors;

- Network bandwidth mainly determined by geographical configuration: There are diverse kinds of network configuration (*NetConf*) such as Wi-Fi and GSM, etc. The networks have their own bandwidth, which is represented with *Bandwidth (NetConf)*.
- Size of dataset flown over the network: The larger amount of dataset requires more time to transmit the dataset. Let *Size(D)* be a size of the dataset transmitted over the network. Hence, it is essential to design the client application and services in a way that *Size(D)* is minimized.
- Number of messages exchanged over the network: To carry out a business process, a number of methods and services can typically be invoked. Let *NumOfMsg(D)* be the number of messages between a mobile system and a service provider. Hence, it is desirable to design the workflow in a way that *NumOfMsg(D)* is minimized.

Let *Cost_to_Communicate(D, NetConf)* be the network communication cost of delivering data, D, while F_{client} invokes F_{server} over the network configuration *NetConf*. Hence, the value of *Cost_to_Communicate(D, NetConf)* is determined;

$$Cost\_to\_Communicate\ (D, NetConf) \propto Size(D) * NumOfMsg(D) * 1/Bandwidth(NetConf). \quad (5)$$

Due to the partitioning in balanced MVC architecture, there exist subsidiary considerations that can affect time efficiency as shown in **Fig. 4**.

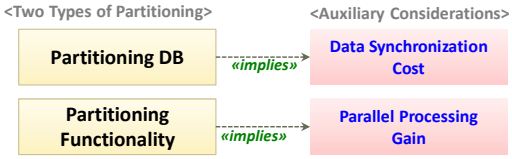


Fig. 4. Subsidiary Factors for *Time Behavior*

Factor 3. Data Synchronization Cost: According to five options of balanced MVC architecture, a portion of the data is replicated to the client side to reduce the number of invocation through the network. Due to the replicated data, $D_{duplicated}$, there is a high risk to violate data integrity. Hence, the amount of performance loss to synchronize data of both sides is affected by the following factors;

- Complexity of synchronization mechanism (F_{syn}): The task of synchronizing states of the replicated data is performed either in the foreground or in the background. To do this, most of the synchronization tasks require thread mechanisms which require additional functionality complexity.
- Number of messages exchanged over the network: Different synchronization mechanisms utilize different communication patterns such as pulling, pushing, or observer pattern. These different patterns affect the number of message passings between client and server sides, $NumOfMsg(D_{duplicated})$.
- Size of duplicated data: A larger size of the data indicates there are many statues to maintain data integrity. Hence, the size of the replicated data (i.e. $Size(D_{duplicated})$) affects the cost of synchronizing data.
- Frequency of data synchronization: Some applications specify that all the data should be synchronized whenever data are modified, and others require that data are synchronized periodically. These different requirements can affect the frequency of data synchronization, $SyncFreq(D_{duplicated})$, which finally results in different response time.

Let $Cost\_to\_Synchronize\ (D_{duplicated})$ be the additional cost to maintain synchronization of data, $D_{duplicated}$. This equation is much similar to $Cost\_to\_Communicate\ (D, NetConf)$ since these two equations deal with additional cost to transfer data over the network, except for $SyncFreq(D_{duplicated})$. Hence, the value of $Cost\_to\_Synchronize\ (D_{duplicated})$ is determined;

$$Cost\_to\_Synchronize(D_{duplicated}) \propto Cost\_to\_Compute(F_{syn}) * NumOfMsg(D_{duplicated}) * Size(D_{duplicated}) * 1/Bandwidth(NetConf) * SyncFreq(D_{duplicated}). \quad (6)$$

This cost is only required when the requirement specifies that the duplicated data should be synchronized right after data are modified, which corresponds to patterns #4 and #5.

Factor #4. Parallel Processing Gain: Due to the intrinsic characteristics of balanced MVC mobile applications, there is a good potential of running functionality in parallel. This could lead the overall performance gain. However, to make functionalities preform in parallel,

supporting mechanism such as thread manipulation is required. Moreover, some data needs to be transferred.

Let *Cost_to_Parallism (ParF,ParD)* be the cost for executing a certain functionality F in parallel. Here *ParF* is a functionality performed in parallel and a part of F_{client} and F_{server} in the equation (1). Hence, *Cost_to_Parallism (ParF,ParD)* is determined;

$$\begin{aligned} &Cost\_to\_Parallism\ (ParF,ParD) \propto \\ &(Cost\_to\_Thread() + Cost\_to\_Communicate\ (ParD,\ NetConf) \\ &- Min\ (Cost\_to\_Compute\ (ParF_{provider}, ParD_{provider}), \\ &Cost\_to\_Compute(ParF_{client},\ ParD_{client}))) \end{aligned} \tag{7}$$

Note that the minimum value between *Cost_to_Compute* $(F_{provider}, D_{provider})$ and *Cost_to_Compute*(F_{client}, D_{client}) is substracted since the two computation costs are already considered in equation (1) and the mininum value is not needed to the total time behavior.

By considering all these time behavior effects of balanced MVC architecture, we can calculate the total value of time behavior for a given balanced MVC mobile application as followings;

$$\begin{aligned} &TotalTimeBehavior \\ &= Cost\_to\_Compute(F_{client},\ D_{client}) + Cost\_to\_Compute(F_{provider},\ D_{provider}) \\ &+ Cost\_to\_Communicate\ (D,\ NetConf) + Cost\_to\_Synchornize\ (D_{duplicated}) \\ &- Cost\_to\_Parallelism\ (ParF,ParD). \end{aligned} \tag{8}$$

Hence, it is desirable to design balanced-MVC mobile application which reveals the lower network communication overhead, the lower object synchronization overhead, and the higher parallelism. Hence, as *TotalTimeBehavior* has a lower value, we can get a better design of balanced MVC architecture.

4.2 Resource Utilization

It is natural that users only use their mobile devices for a very limited time if they run large-scaled mobile applications on mobile devices. For example, when your mobile devices use GPS sensors for a long time, you can easily notice that the battery is quickly drained. Hence, we drive *resource utilization* as the second criterion. Although ISO/IEC defines external and internal metrics for resource utilization, they are genetically defined without considering unique characteristics of mobile applications. We now define two factors which have a tremendous impact on the resource utilization.

Factor 1. Memory Consumption: The drain of memory results in executing functionalities slowly, even mobile application can suddenly stop. Hence, the amount of memory consumption affects overall efficiency of mobile applications. Through our experiences where we face memory drain problems in developing mobile applications, we identify that the following two factors affect the degree of memory consumption.

- Size of data in the memory $Size(D_{memory})$: Typically, datatype with larger size occupy larger memory space.
- Size of instances in the memory $(Size(Instance_{memroy}))$: Similarly, a fact that there are many instances in the memory indicates that the memory is occupied by the instances. This value is determined by the number of instances and the size of each instance. Here, each size of the instance is affected by the complexity of the application.

With these factors, we can evaluate that the degree of memory consumption, *Memory ( )*, as following;

$$Memory\ (\ ) \propto (MemAllocation\ (Size(D_{memory})) + (MemAllocation(Size(Instance_{memory}))\ ) * 1/\ total\ memory\ capacity \quad (9)$$

This value depends on the total capacity of the memory allocation, *MemAllocation( )*.

Factor 2. Battery Consumption: Among resources of mobile devices, a battery is the most sensitive resource to using mobile applications. It means that without the battery, we cannot use any mobile application, and the degree of battery consumption can determine the use of mobile application. Through our experiences, we conclude that the following three factors affect the degree of battery consumption.

- The functionality complexity of a mobile application (*F*): It is natural that running applications with high functional complexity consumes too much battery.
- The size of data required by a mobile application (D): When application manipulates dataset by accessing DB, it consumes resources. The consumption depends on the amount of dataset.
- The frequency of using network: Network communication requires some energy which is provided by the battery. Hence, the more frequently data are flown over the network, the more quickly the battery is drained.
- The frequency of using sensors (*Freq(Sensor$_i$)*): Similar to the network, the usage of sensor also requires consuming the battery. Since each sensor, such as GPS and proximity sensor, consumes different amount of battery, we need to consider the summation of (*Freq(Sensor$_i$)* for all types of sensors.

With these factors, we can evaluate the degree of battery consumption, *Battery( )*, as following;

$$Battry\ (\ ) \propto BatteryConsumption\ (F) * BatteryConsumption\ (F,D) * BatteryConsumption\ (D,\ NetConf)\ * BatteryConsumption\ (\sum_i(Freq(Sensor_i)) * 1/total\ battery\ capacity \quad (10)$$

Like *Memory ( )*, the value of *Battery ( )* also depends on the total capacity of the battery consumption, *BatteryConsjumption( )*. And, since the degree of battery consumption is related to *Cost_to_Compute (F,D)* and *Cost_to_Compute(F)*, a design with low values of them can lead to a low value of the degree of battery consumption.

By considering all these resource utilization effects of balanced MVC architecture, we can calculate the total value of resource consumption for a given balanced MVC mobile application as followings;

$$TotalResourceConsumption \propto (Memory\ (\ ) + Battery\ (\ ).) \quad (11)$$

Hence, it is desirable to design balanced-MVC mobile application which reveals the lower amount of memory consumption and lower amount of battery consumption. Hence, as *TotalResourceConsumption* has a lower value, we can get a better design of balanced MVC architecture.

5. Instructions for Applying Balanced MVC Architecture

Although balanced MVC architecture is devised for accommodating highly complex applications on mobile devices, there is a performance penalty due to functionality partitioning. Hence, we should design a right form of balanced MVC architecture which maximizes its strong point and minimizes the performance overhead in terms of *time behavior* and *resource utilization*. **Fig. 5** illustrates the relationships between two types of efficiency factors and the

overall quality of architectures. An *x-axis* indicates the amount of thge resource consumption which is returned from the equation (8), and a *y-axis* indicates the length of response time which is acquired from the equation (11).

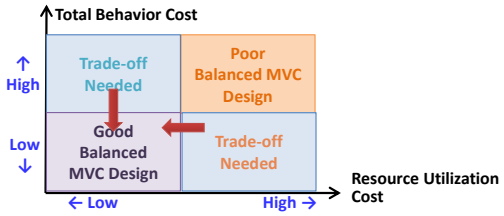


Fig. 5. Criteria for balanced MVC architecture design

The designed balanced MVC architecture is acceptable when *x-axis* and *y-axis* have lower values. These are the key design principles for balanced MVC architecture, which is also utilized in evaluating the architecture.

5.1 Process to Design Balanced MVC Architecture

In this section, we devise a method to design the most appropriate balanced MVC architecture for the target mobile application.

Designing a balanced MVC architecture begins with three input artifacts 오류! 참조 원본을 찾을 수 없습니다.; *requirement specification* for the target mobile application, *use case model*, and *object model*. By using these artifacts, we suggest the following three-step process by considering characteristics of five patterns;

- Determine whether client application manages its own DB or not.
- Determine whether classes in control layer need to be partitioned to both sides or not.
- Refine the architecture by reconsidering *time behavior* and *resource utilization*.

Step 1: We clarify whether there is a need that for client applications to maintain their own database. That is, the need of *C.DB* and also *C.Model* is determined. This decision is made by considering *requirement specification*.

After this decision, *C.Model* and *S.Model*, which correspond to *C.DB* and *S.DB*, need to be shaped. At this time, we utilize the previous artifacts such as *use case model* and *object model*. Classes in the class diagram mostly manage the behavior and data of the target application, which is a key input for this step. All the classes for the server side derive classes in *S.Model*, which results in shaping dataset in *S.DB*. And, we also apply the following guideline to derive *S.Model*.

G1. Place dataset on the server side (i.e. S.DB) if the amount of data manipulated is too large for client application to handle. Due to resource limitation on mobile devices, they cannot hold large amount of dataset. Hence, the dataset is managed in server side, S.*DB*.

However, classes in *C.Model* should be derived by considering *time behavior* and *resource utilization* since mobile devices cannot hold the large amount of dataset like *S.DB*. For this, we apply the following guideline.

G2. Place the minimum set of dataset on the client side (i.e. C.DB) by considering dependency on functionality and security and privacy issues. If there is the most frequently used functionality by users, dataset relevant to this functionality needs to be placed to *C.DB*. And also, privacy-sensitive dataset should be maintained in the client side.

G3. By considering the size of duplicated data, readjust the location of C.DB. This guideline is related to $Size(D_{duplicated})$. The larger amount of data C.DB is managed, the larger

amount of cost data synchronization requires. That is, if we reduce the value of $Size(D_{duplicated})$, the overhead cost can be minimized.

Step 2: The most important decision is to design classes in the *C.Control* and *S.Control* since this decision affects overall efficiency severely. That is why most of the control classes perform business processes which can yield larger amount of network cost by interacting with more than one model classes. Hence, we define classes in *C.Control* and *S.Control* by considering *time behavior* and *resource utilization*.

Both quality criteria are influenced by a same set of factors including complexity of the functionality (F), the size of dataset (D), the size of dataset flown over the network, the number of messages exchanged over the network, and others. By considering all these factors, we define the following guidelines to derive control classes.

G4. If the number of message flows over the network is intensive, consider placing the related functionalities on one side. This guideline concerns *NumOfMsg(D)* and affects *Cost_to_Communicate ( )* and *Battery ( )*. Hence, to reduce them, control classes and relevant model classes should be in the one side.

G5. Choose the appropriate synchronization mechanism which will be located in C.Control and S.Control. Duplicating part of *S.DB* to *C.DB* requires data synchronization. As shown in equation (4), well-designed synchronization mechanism can reduce the amount of synchronized data, the frequency of data synchronization, and the complexity of the synchronization method.

With the guideline G4 and G5, the most appropriate pattern is decided. After that, we should extract the right classes for *C.Control* and *S.Control*. Since a main role of control classes is to mediate interactions among multiple model classes, it is recommended that a control class performs a set of similar business processes. Hence, our method is based on functionality grouping, which may be reflected in use case model. These are commonly applied criteria to grouping functionalities;

- *Sub-system (Functionality)*: Different controllers manage different types of functionalities.
- *Access Authority*: According to the types of users (s.a. manager and member), they have different authority to invoke functionality.
- *Location (Zone)*: According to the locations, users may have limitation on invoking certain functionality. For example, some are not accessible outside their workplace.

We figure out functionality groups by examining use case model. Then, we refine the functional groups by considering efficiency or functionality similarity. And, we define control classes in a way that a control class is derived from a refined functionality group. At this time, by checking what classes are located in client and server sides, classes in *C.Cotrol* and *S.Control* are extracted by following guidelnes.

G6. Locate functionality with high complexity on server side. This guideline is related to reducing all the metrics in (6), (7), and (8). Running functionality with high complexity tends to spend a large amount of time and consume high resources of CPU and Memory.

G7. Locate functionality with a large amount of data manipulation on server side. This guideline concerns *Size(D)* and affects all the metrics in (6), (7), and (8). Hence, to reduce the values in these metrics, such functionality is located on the *S.Control*.

By performing Step 1 and Step 2, mobile application architecture with the appropriate pattern is designed. **Fig. 6** shows an overall process to integrate the two steps and applicable guidelines.

The process begins with checking whether there is a separate object model for client system. If there is no the separate object model, one of the *Pattern 1*, *Pattern 2*, and *Pattern 3* can be chosen. Otherwise, either *Pattern 4* or *Pattern 5* is chosen for the target application.

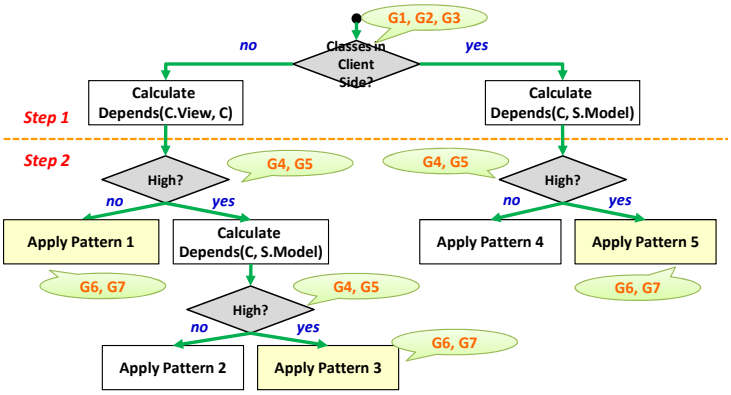


Fig. 6. Process to Apply Architectural Patterns

Once applicable patterns are chosen, we should decide whether control layer *C* is partitioned to both sides or not by considering the degree of dependency among components. The dependency is represented with a function *Depends(A, B)* that returns the estimated degree of dependency between *A* and *B* and the range of the value is between 0 and 1. *Depends( )* is acquired by considering *Size(D)*, *NumOfMsg(D)*, and other communication related factors. That is, Depends ( ) is quite relevant to *Cost_to_Communicate(D, NetConf)*, *Cost_to_Synchronize* ($D_{duplicated}$), and *Battery ( )*. For the patterns #1, #3, and #5, we decide functionalities with high complexity are located to *S.Control* to reduce values of *Cost_to_Compute ( )* and *Memory ( )*.

Step 3: In the previous steps, we applied appropriate patterns to the target mobile application without considering *Cost_to_Parallelism ( )*. This step is to refine the architecture by reconsidering *time behavior* and *resource utilization* in order to define the most optimal architecture to the application.

G8. Place functionalities that can be processed in parallel on both client and server sides, i.e. C.Control and S.Control. This guideline is related to *Cost_to_Parallelism( )*. We can get performance gain by applying parallel processing mechanisms. Hence, among the classes in *C.Control* and *S.Control*, we define what classes can be executed in parallel.

G9. By considering the complexity of thread handling mechanism and degree of parallel processing, readjust the functionality partitioning. This guideline is related to *Cost_to_Parallelism( )*Thread handling mechanism is essentially required for parallel processing, but could increase functional complexity. If the overhead is too high to hamper the benefits of parallelism, we examine the initial design on functionality distribution.

5.2 Evaluating Balanced MVC Architecture

Since software architectures are highly conceptual artifacts, there is no way of running and monitoring their quality. On the other hand, we wish to evaluate the designed the architecture before realizing the architecture into its implementation. Hence, we propose a four-step scenario-based **오류! 참조 원본을 찾을 수 없습니다.** and simulation-based evaluation method. There are two underlying reasons for our approach.

- Due to complex runtime environment of mobile applications, it is challenging to evaluate those mobile applications before actual implantation. Hence, we adopt *simulation*-based evaluation to quantitatively assess architecture.
- To simulate the mobile applications, it is necessary to determine baseline values of each operand in the metrics. Without actually running the mobile applications, it is somewhat hard to predict them. To help it, we adopt *scenario*-based evaluation.

Our method is to evaluate the architecture in terms of two key criteria defined in section 4; *time behavior* and *resource utilization*.

Step 1 is to define key scenarios from use case model by adopting scenario-based approach. The purpose of this step is to understand the target application to be evaluated in terms of all the functionalities and their sequences.

Step 2 is to determine the values of terms used in the metrics used in section 4, which is used for simulation. The ultimate purpose of the evaluation is to acquire an expected total cost, *TotalTimeBehavior* in equation (8) and *TotalResourceConsumption* in equation (11) by considering all the scenarios. First, drawing from the scenarios, we need to acquire values of the terms in metric as shown in **Table 1**.

Table 1. Acquiring Values of Terms used in Metrics

Sub-Equation	Variables	How to Acquire Values of Terms in Metrics
Cost_to_Compute_without_DB (F)	*Size(InParam(F))*	A proportional number by comparing the size of input parameters such as primitive and abstract types
	NumOp (F)	An actual number of operations for performing the functionality, which is described in the scenarios
Cost_to_Compute_with_DB (F, D)	*Size(D)*	A proportional number by comparing actual size of data such as primitive and abstract data types
	Complexity (QueryOp(D)	A proportional number by comparing types of queries. Typically, a search query is more expensive than the other queries.
	Freq(QueryOp(D))	The number of invoking queires in the scenarios
Cost_to_Communicate (D, NetConf)	*Bandwidth (NetConf)*	A proportional number by comparing actual bandwidth for the given bandwidth
	Size(D)	A proportional number by comparing actual size of data such as primitive and abstract data types
	NumOfMsg(D)	An actual number of data exchanges described in the scenarios
Cost_to_Synchronize (D)	Same as *Cost_to_Communicate (D, NetConf)*	
Cost_to_Parallelism (F,D)	Same as *Cost_to_Compute_with_DB(F, D)*	

Similarly, in equation (11), there are two variables; *Memory( )* and *Battery( )*, which are determined by equation (9) and equation (10) respectively. For evaluation equation (9), $Size(D_{memory})$ is expected by adding actual sizes of all D_{memory} whether D_{memory} is primitive or abstract data type. $Size(Instance_{memory})$ is also expected by considering all class in memory at the same time, which adds results of multiplying the possible number of $Instance_{memory}$ and the size of $Instance_{memor}$ for all $Instance_{memory.}$ For evaluation equation (10), $Freq(Sensor_i)$ is expected by calculating the number of using sensors in one use case.

It is challenging to evaluate the resulting architecture on runtime environment. That is why the data capability of devices and network situation such as bandwidth are continuously

evolving, and those environment parameters are not known. Hence, these dynamic aspects should be tested with architecture simulation tools, which can take sets of environmental parameters generated with probability-based random generators such as Poisson Distributor, as shown below. That is, the resulting architecture can be simulated and tested with different sets of values for the terms used in equations (1) through (11).

Step 3 is to measure the total costs of a target mobile application with balanced MVC architecture, with a number of different value sets for different design options and patterns. For the designed balanced MVC architecture, we perform at least five simulations since there are five applicable patterns. One simulation is for the architecture with the determined pattern, and the others are for the rest of patterns which are not chosen.

Step 4 is to compare the total costs of multiple simulations and to re-design the architecture if the expectation is not fulfilled through the evaluation. Drawing from the simulations, we compare the results and check whether *TotalTimeBehavior* and *TotalResourceConsumption* are the lowest value among the five patterns.

6. Case Studies

6.1 Case Study of Applying Balanced MVC

We have applied the proposed balanced MVC in developing a commercial Android application, Mobile Mate Service (MMS), which provides a location-based social networking service. With this case study, we show how the proposed architecture can be applied in practice, and discuss key benefits resulted from the architecture.

6.1.1 Functionality

MMS provides four groups of functionality; *Profile Management* is to manage users' profile such as privacy setting and hobby, *Membership Management* is to manage membership information, *Group Management* is to manage registered members with groups, and *Location Service* is to display locations of members and to compute the expected travel time.

A typical operation scenario with MMS is for a user to register the membership, to invite colleagues into groups maintained by the user, and to accept invitations for joining groups, and to observe the locations of colleagues with Google Map and estimated travel time to destinations, as shown in **Fig. 7**.

Fig. 7. Locations of Colleagues and current User are displayed

6.1.2 Architecture

There are a number of challenging functional and extra-functional requirement items with MMS. One of them is to handle long-lasting transaction among members when a member invites another member of mobile device is momentarily turned off. To meet the challenges, we rely on the architecture design.

Our use case model for MMS includes 24 use cases, and many of them require both client-side and server-side functionalities. By applying the typical architecture design process such as [18], balanced MVC was adopted as the most optimal scheme. Among the five patterns of balanced MVC as presented in section 3.2, the pattern #5 was adopted to handle the high dependency between *C.Control* and *S.Model*. The resulting functional view of the architecture is shown in **Fig. 8**.

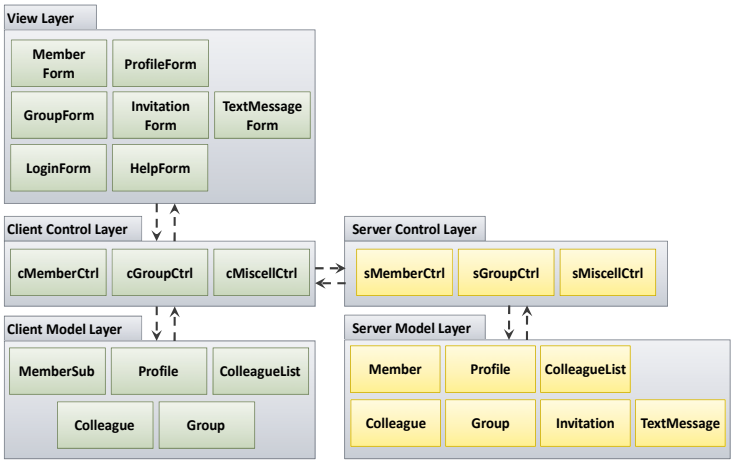


Fig. 8. Functional View of MMS Architecture with Pattern #5

There are five layers in the resulting functional view, and pattern #5 specifies four interaction paths as shown in the figure. The interaction paths are consistently reflected in our sequence diagrams as show in **Fig. 9**. As shown, objects on client sides are interacting with objects on server side according to the paths allowed in the pattern 5.

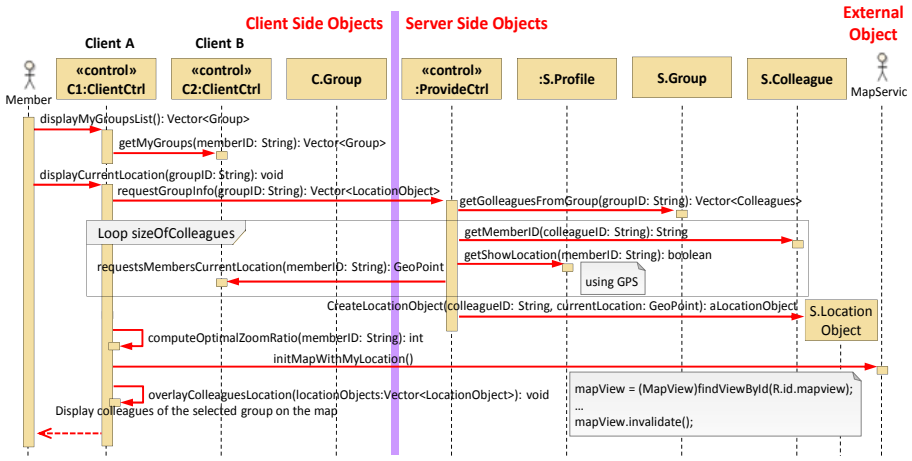


Fig. 9. Sequence Diagram showing Interaction Paths

6.1.3 Evaluation of Resutling Architecture

The functional and extra-functional requirements for MMS could not be realized with conventional MVC. In our implementation of MMS with balanced MVC, we could achieve the following benefits;

- On the functionality and datasets specific to client side are allocated to *C.Control* and *C.Model*. Consequently, the thin-client with minimal functionality was implemented.
- Server side layers and components provide inter-user functionality and long-lasting transactions such as inviting member and accepting invitation. This type of functionality could not be implementable and runnable on mobile devices due to the resource constraints and nature of the functionality.
- Server side dataset in *S.Model* layer provides persistency on datasets, even if datasets on mobile device get lost. That is, data integrity and persistency were enhanced with balanced MVC.
- On each side, the partitioning of functionality into *Control layer* and *Model layer* was well adopted. That is, all the underlying principles and benefits of MVC were still realized in balanced MVC.

6.2 Case Study of Comparing 5 Patterns

We have applied the proposed architecture evaluation scheme with a mobile application providing sorting functionality. With this case study, we show how the proposed architecture can be evaluated and claim that the most optimal architecture design is essential in determining the overall quality of the target system.

6.2.1 Target Mobile App and Experiment Settings

We implemented an Android application, *Sorting Application*, which runs two sorting algorithms; *Merge sort* and *Selection sort* which have different levels of complexity, $O(n \log n)$ and $O(n^2)$, respectively. The application retrieves a number of integer values from a file stored on hard disk. We have run the application with two scenarios which are different in terms of *functionality complexity*, *data capacity*, and *interaction intensity*. The level of functionality complexity is set with the number of repeatedly performing each soring algorithm within an invocation and the number of integers to be sorted. And, the degree of interaction intensity is the number of interactions between user and control layers while one user request is processed. We project variations to define three scenarios as shown in **Table 2.** *Scenario #1* represents a situation of having two functionalities with different degree of the complexity, *Scenario #2* represents a situation of managing different capacities of the database, and *Situation #3* represent a situation of having different degrees of interaction intensity.

Table 2. Condition for Scenarios

Scenario / Conditions	#1	#2	#3
Functionality 1 (Merge Sort)			
The number of performing sorting algorithm per an invocation	1	10	10
The number of integers to be sorted	10	100	10
The number of intergers stored in databse	100,000	100,000	100,000
The number of interactions between view and control layers	10	10	10
Functionality 2 (Selection Sort)			
The number of performing sorting algorithm per an invocation	100	10	10
The number of integers to be sorted	50	100	10
The number of intergers stored in databse	100,000	100	100,000
The number of interactions between view and control layers	10	10	30

6.2.2 Interpretation of the Case Study

With the different scenarios, we measure *response time* for evaluating time efficiency, and *memory* and *battery consumption* for evaluating resource efficiency.

In case of *scenario #1,* the application does not manage such large amount of the data in the client application by applying *G1*. As a result of Step 1, pattern 1, 2, or 3 is the candidate of this scenario. In the step 2, we try to apply guidelines from *G4* to *G7*. By applying *G4*, we decide that all the control classes can be located either *C.Control* or *S.Control* due to lower interactions between view and control layers. That is, the interaction intensity does not affect any decision in this scenario. And, *G5* is not applied since there is not C.DB in this scenario. Finally, since a functionality of the selection sort is large, that functionality is located on the server side by applying *G6* and *G7*. A functionality of merge sort is comparatively low, that functionality is located on the client side.

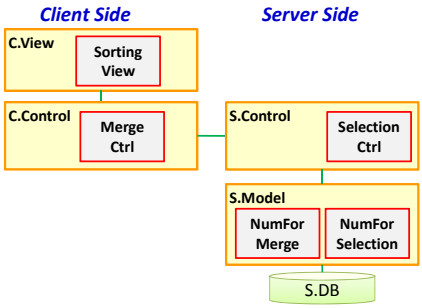


Fig. 10. Applying Pattern #3 to Scenario #1

As a result, *Pattern 3* turns out to be the most appropriate pattern for this scenario as shown in **Fig. 10**. With the architecture, we estimate the value of time behavior by using equation (4). For the comparison, we also measure time behavior of the applications applying pattern 1 and 3. **Table 3** shows an evaluation result which has a proportional number since it is hard to compute an abstract value for each factor. The values are acquired by using guidelines in **Table 1**. Note that these values are subjectively determined, rather than objectively. With this evaluation result, we can check pattern 3 is the most appropriate to this scenario.

Table 3. Evaluation Result of Scenario #8

Factors \ Pattern	#1	#2	#3
Cost_to_Compute (F_{client}, D_{client})	0	1.8	0.2
Cost_to_Compute (F_{server}, D_{server})	0.8	0	0.3
Cost_to_Communicate (D, NetConf)	0.5	1.2	0.3
Result	1.3	3.0	0.8

To check whether *Pattern 3* has higher efficiency for this scenario, we developed the application and measured efficiency. **Fig. 11** shows the result of measuring time and resource efficiency for scenario #1. With time and resource efficiency viewpoints, *Pattern 3* has the lowest value. *Pattern 1* consumes a large amount of network communication cost due to interaction between view and control layers while its computation cost is quite low. *Pattern 2* also consumes an extremely large amount of network communication cost since the client

application requires integers to be sorted from *S.DB* and also computation cost is quite high. Compared to these patterns, *Pattern 3* consumes larger computation cost than *Pattern 1* and lower computation cost than *Pattern 2*. Pattern 3 consumes similar amount of network communication cost to *Pattern 1* and much lower network communication cost than *Pattern 2*. Hence, we can conclude that *Pattern 3* is most efficient for this scenario.

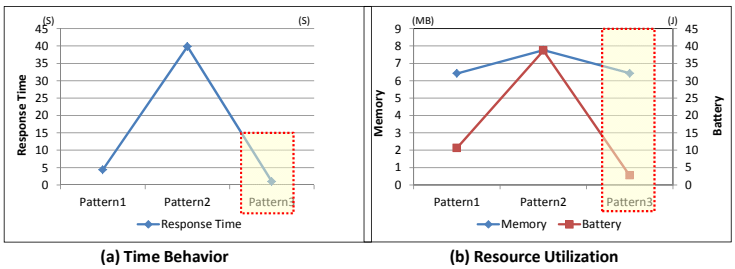


Fig. 11. Results for Scenario #1

In case of *scenario #2,* the application can manage dataset both client and server sides due to different database capacity by applying *G1*. In the step 2, we try to apply guidelines from *G4* to *G7*. By applying *G4*, we decide the location of control classes which are the same location as the model layer. For the simplicity, we do not consider synchronization and parallel processing in this scenario. As a result, *Pattern 5* turns out to be the most appropriate pattern for this scenario as shown in **Fig. 12**.

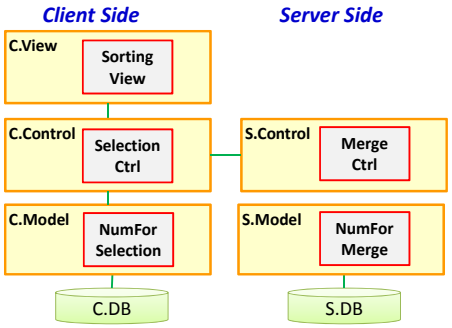
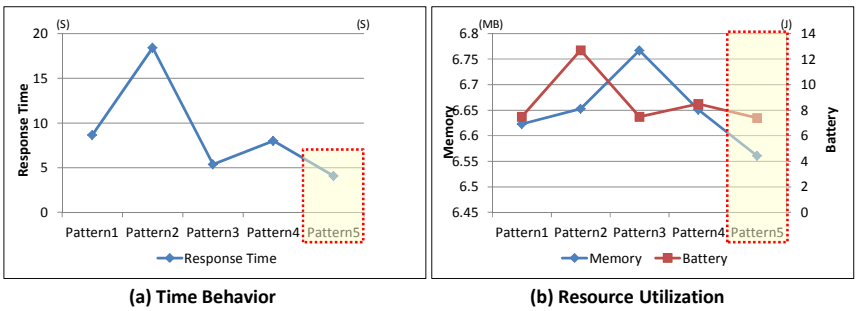


Fig. 12. Applying Pattern #5 to Scenario #2

In the similar way of scenario #1, we also check whether pattern #5 is the most efficient by applying the evaluation method in the section 5.2, which corresponds to our assumption. To check whether *Pattern 5* has higher efficiency for this scenario, we developed the application and measured efficiency. **Fig. 13** shows the result of measuring time and resource efficiency for scenario #2.



(a) Time Behavior **(b) Resource Utilization**

Fig. 13. Results for Scenario #2

Pattern 5 has the lowest values of response time and resource utilization. Pattern 5 has larger computation cost than Pattern 1, similar computation cost to Pattern 3, and smaller computation cost than Pattern 2 and Pattern 4. And, *Pattern 5* is the lowest network communication cost. Hence, *Pattern 5* is turned out to be the most efficient pattern for this situation.

In case of *scenario #3,* the application does not manage such large amount of the data in the client application by applying *G1*. As a result of Step 1, pattern #1, #2, or #3 is the candidate of this scenario. In the step 2, we try to apply guidelines from *G4* to *G7*. By applying *G4*, we decide that all the control classes are located to *C.Control* due to frequent interactions between actor and application. Otherwise, *NumOfMsg (D)* is extremely increased. And, *G5* is not applied since there is not C.DB in this scenario. Finally, due to the low functionality complexity, *G6* and *G7* are not also applied. As a result, *Pattern #2* turns out to be the most appropriate pattern for this scenario as shown in **Fig. 14**.

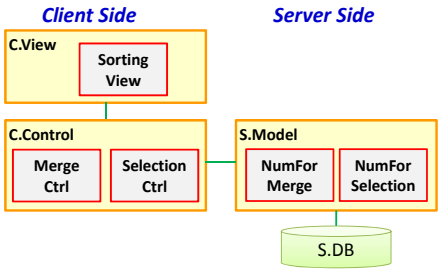


Fig. 14. Applying Pattern #2 to Scenario #3

Similarly, we also check whether pattern #2 is the most efficient by applying the evaluation method in the section 5.2, which corresponds to our assumption. To check whether *Pattern 2* has higher efficiency for this scenario, we developed the application and measured efficiency. **Fig. 15** shows the result of measuring time and resource efficiency for scenario #2. *Pattern 2* has the lowest values of response time. As the candidates, *Pattern 1* and *Pattern 3*, they have lower efficiency since they yield more network overheads due to frequent interactions between actor and application.

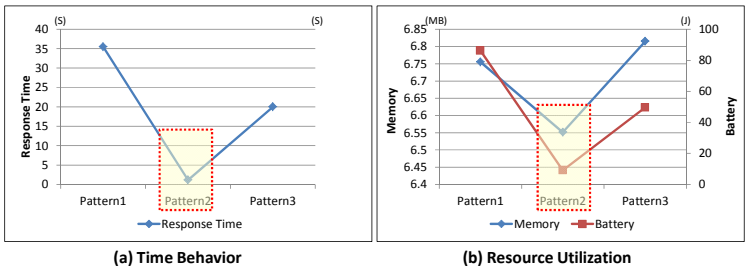


Fig. 15. Results for Scenario #3

6.3 Case Study of Comparing Balanced MVC to Fat Client and Fat Server

In this section, we perform another case study to prove whether our patterns can ensure better efficiency than other conventional architecture patterns. There are a number of architecture patterns/styles known such as shared repository, client-server architecture [18]. Since the proposed patterns deal with functionality partition, we select patterns having the same purpose; *Fat Client (i.e. Standalone) Pattern* and *Fat Server (i.e. Mobile Web) Pattern*. For this, we implement two additional versions of sorting application for the Scenario #2 as shown in **Fig. 16.** And, for the balanced MVC architecture, we utilize the architecture in **Fig. 12.**

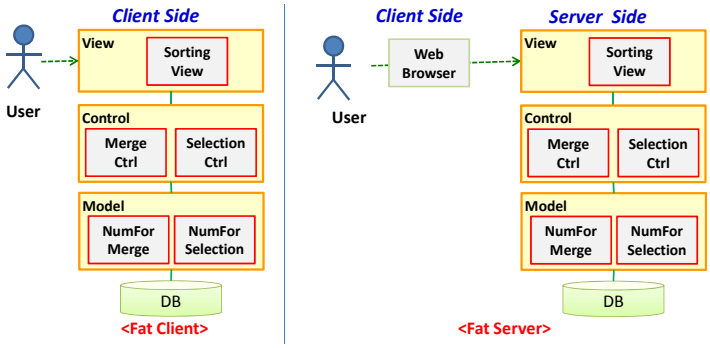


Fig. 16. Architecture with Fat Client and Fat Server

For these three different architecture designs, we measure *response time* for evaluating time efficiency, and *memory* and *battery consumption* for evaluating resource efficiency. **Table 4** shows the result of performing this case study.

Table 4. Evaluation Result of Scenario #8

QoS / Pattern	Response Time (ms)	Memory Usage (KB)	Battery Consumption (J)
Fat Client	15.102	9.588	23.1
Fat Server	4.332	6.22	8.9
Balanced MVC	4.073	7.4	6.561

In case of *Fat Client pattern*, although no network communication is required, the largest response time is consumed since all the functionalities with substantial complexity should be

run on mobile device. Correspondingly, a large amount of resources is consumed. In addition, *Fat client* has a severe limitation on managing extremely large amount of dataset.

In case of *Fat Server pattern*, computation time is less consumed than the one of balanced MVC architecture, but it consumes larger amount of network communication time. As a result, the overall response time is similar to balanced MVC architecture. However, more resources are consumed in *Fat Server* since the application needs to wait for a reply from sever application while keeping its network connection. Although we do not consider parallel processing in this case stud, we can ensure that balanced MVC architecture can yield much better efficiency than *Fat Server* if some functionalities are processed in parallel.

7. Conclusion

Due to limited computing power and hardware resource of mobile devices, large-scaled applications could not be deployed on these devices. To remedy the limitations in terms of performance, it is inevitable to let some heavy-weight functionality executed on the server side and let client application invoke the functionality in the server. To realize this kind of mobile applications, we proposed a unique, ideal and practical architecture for mobile applications, called *balanced MVC architecture*.

To design balanced MVC architecture for a target mobile application by considering performance efficiency, we define key two criteria by adopting ISO/IEC 9126; *time behavior* and *resource utilization*. And, by considering the criteria, we define a three-step method to design a balanced MVC architecture which embodies functionality partitioning for high performance. To make sure whether a designed architecture is fully satisfied with the criteria, we presented a four-step method to evaluate the balanced MVC architecture. The evaluation method is based on a simulation which calculates an expected the value of total performance by using use case scenarios. Finally, we performed three case studies to show how the architecture is applied in practice, prove how correctly the architecture is evaluated, to compare our architecture patterns with other conventional ones. As our future research, we are developing simulation-based evaluation tool. The experiment results are used to set up baseline values for calculating all the equations defined in this paper.

References

[1] B. König-Ries and F. Jena, “Challenges in mobile application development,” *it-Information Technology*, vol.52, no.2, pp.69-71, 2009. Article (CrossRefLink).

[2] G.H. Forman and J. Zahorjan, “The challenges of mobile computing,” *IEEE Computer*, vol.27, no.4, pp.38-47, Apr.1994. Article (CrossRefLink).

[3] ISO/IEC, *ISO-IEC 9126-1 Software Engineering – Product Quality – Part 1: Quality Model*, 2001.

[4] R. Tergujeff, J. Haajanen, J. Leppanen and S. Toivonen, “Mobile SOA: Service orientation on lightweight mobile devices,” in *Proc. of 2007 IEEE Int. Conf. on Web Services*, pp.1224-1225, Jul.2007. Article (CrossRefLink).

[5] Y. Natchetoi, V. Kaufman and A. Shapiro, “Service-oriented architecture for mobile applications,” in *Proc. of the 1st Inernational Workshop on Software Architectures and Mobility*, pp.27-32, May. 2008. Article (CrossRefLink).

[6] A. Ennai and S. Bose, “MobileSOA: A service oriented web 2.0 framework for context-aware, lightweight and flexible mobile applications,” in *Proc. of the 2009 12th Enterprise Distributed Object Computing Conf. Workshop*, pp.348-382, Sept.2008. Article (CrossRefLink).

[7] K. Kumar and Y.H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *IEEE Computer*, vol.43, no.4, pp.51-56, Apr.2010. Article (CrossRefLink).

[8] M. Hassan, W. Zhao and J. Yang, "Provisioning web services from resource constrained mobile devices," in *Proc. of 2010 IEEE 3rd International Conference on Cloud Computing*, pp. 490-497, 2010, Article (CrossRefLink).

[9] M.T. Tran, Y.H. Kim and J.H. Lee, "Load balancing and mobility management in multi-homed wireless mesh networks," *KSII Transactions on Internet and Information Systems*, vol.5, no.5, pp.959-975, May.2011, Article (CrossRefLink).

[10] R. Matero and J.W. Lee, "Dynamic service assignment based on proportional ordering for the adaptive resource management of cloud systems," *KSII Transactions on Internet and Information Systems*, vol.5, no.12, pp.2294-2314, Dec.2011, Article (CrossRefLink)

[11] C. Ryan and P. Rossi, "Software, performance, and resource utilisation metrics for context-aware mobile applications," in *Proc. of the 11th IEEE International Software Metrics Symposium*, pp.12, Sept.2005. Article (CrossRefLink).

[12] F. Aquilani, S. Balsamo and P. Inverardi, "Performance analysis at the software architectural design level," *Performance Evaluation*, vol.45, no.2-3, pp.147-178, 2001. Article (CrossRefLink).

[13] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen and O. Salo, "Mobile-D: An agile approach for mobile application development," in *Proc. of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications,* pp.174-175, Oct.2004. Article (CrossRefLink).

[14] V. Rahimian and R. Ramsin, "Designing an agile methodology for mobile software development: A hybrid method engineering approach," in *Proc. of the 2nd International Conference on Research Challenges in Information Science*, pp.337-342, Jun.2008. Article (CrossRefLink).

[15] J. Häkkilä and J. Mäntyjärvi, "Developing design guidelines for context-aware mobile applications," in *Proc. of the 3rd International Conference on Mobile Technology, Applications, and System,* pp.1-7, Oct.2006. Article (CrossRefLink).

[16] M. Sá and L. Carriço, "Lessons from early stages design of mobile applications," in *Proc. of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services*, pp.127-136, Sept.2008. Article (CrossRefLink).

[17] N.Z. Ayob, R.C. Hussin and H.M. Dahlan, "Three layers design guideline for mobile application," in *Proc. of 2009 International Conference on Information Management and Engineering*, pp.427-431, Apr.2009. Article (CrossRefLink).

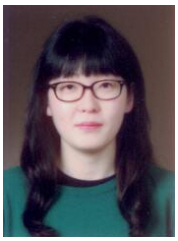
[18] N. Rozanski E. and Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, Addison Wesley, 2005.

[19] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, Wiley, 1996.

[20] J. Kleinberg and E. Tardos, *Algorithm Design.* Addison Wesley 2005.

[21] E.M. Kim, O.B. Chang, S. Kusumoto and T. Kikuno, "Analysis of metrics for object-oriented program complexity," in *Proc. of 8th Annual International on Computer Software and Applications Conference,* pp.201-207, Nov.1994. Article (CrossRefLink).

[22] R. Lee and B. Jeng, "Load-balancing tactics in cloud," in *Proc. of 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery,* pp.447-454, 2011. Article (CrossRefLink).



Hyun Jung La is a research staff in the Services and Software Engineering Laboratory at Soongsil University, Seoul, Korea. She is also a lecturer at the University, offering courses of software engineering and object-oriented analysis and design. She received her master and Ph.D. degrees from Soongsil University in 2005 and 2011 respectively. Dr. La has been actively engaged and played the key role of software architect in large-scaled projects for the past years. Her research interests include cloud services, software architecture design, and advanced mobile computing.

Soo Dong Kim is a professor in the department of Computer Science at Soongsil University, Seoul, Korea. He received his B.S. degree in Computer Science from Northeast Missouri State University in 1984, and his Master and Ph.D. degrees from the University of Iowa, Iowa, USA in 1988 and 1991 respectively. His research interests include software architecture, cloud services, advanced mobile computing, and smart services. He has been actively engaging in large-scaled industry projects, and providing extensive training on software technology to IT industry.