Interactive Mobile Applications Development Using Adapting Component Model

Haeng-Kon Kim and Roger Y. Lee

Abstract In the reason of the variability which characterizes the context of such environments, it is important that mobile applications are developed so that they can dynamically adapt their extra functional behavior, in order to optimize the experience perceived by their users. In this paper we discuss some of the problems of the current mobile based human management applications and show how the introduction of adaptive CBD (Component Based Development) model provides flexible and extensible solutions to mobile applications. Mobile applications resources become encapsulated as components, with well-defined interfaces through which all interactions occur. Builders of components can inherit the interfaces and their implementations, and methods (operations) can be redefined to better suit the component. New characteristics, such as concurrency control and persistence, can be obtained by inheriting from suitable base classes, without necessarily requiring any changes to users of these resources. We describe the mobile applications frameworks and adaptive component model, which we have developed, based upon these ideas, and show, through a prototype implementation, how we have used the model to address the problems of referential integrity and transparent component (resource) migration. We will show the prototyping applications using our approaches. We also give indications of future work.

Keywords Mobile applications · Frameworks · Adaptive component-based development model · Referential integrity · Mobility · Distributed systems

H.-K. Kim (✉)
School of Information Technology, Catholic University of Daegu, Daegu, Korea
e-mail: hangkon@cu.ac.kr

R.Y. Lee (✉)
Department of Computer Science, Central Michigan University, Mount Pleasant, USA
e-mail: lee@cps.cmich.edu

R. Lee (ed.), *Applied Computing and Information Technology*,
Studies in Computational Intelligence 695,
DOI 10.1007/978-3-319-51472-7_5

1 Introduction

Typically, mobile computing is defined as the use of distributed systems, comprising a mixed set of static and mobile clients [1]. More refined and componentized approaches have also been proposed with new mobile development paradigms [2]. It is widely accepted that building in safety early in the development process is more cost-effective and results in more robust design [3]. Safety requirements result from safety analysis—a range of techniques devoted to identifying hazards associated with a system and techniques to eliminate or mitigate them [3, 4]. Safety analysis is conducted throughout the whole development process from the requirements conception phase to decommissioning. However, currently the approaches for incorporating safety analysis into use case modeling are scarce. The Unifying Modeling Language (UML) [5] is gaining increasing popularity and has become de facto industry standard for modeling various systems many of which are safety-critical. UML promotes use case driven development process [5] meaning that use cases are the primary artifacts for establishing the desired behavior of the system, verifying and validating it. Elicitation and integration of safety requirements play a paramount role in development of safety-critical systems. Hence there is a high demand on methods for addressing safety in use case modeling. Naturally, the development of software applications featuring such a sophisticated behavior is not easy. It has been suggested that while researchers have made tremendous progress in almost every aspect of mobile computing, still not enough has been achieved in dealing with the complexity which characterizes their development [6]. We will show how making the change to a component-based Development system can yield an extensible infrastructure that is capable of supporting existing functionality and allows the seamless integration of more complex resources and services. We aim to use proven technical solutions from the distributed component-based Development community to show how many of the current problems with the Web can be addressed within the proposed model. In the next section, a critique of the current Web is presented, highlighting existing problems in serving standard resources and the current approach for incorporating nonstandard resources. The section entitled the mobile applications frameworks and adaptive component model component design, its aims, component model, and system architecture. The Illustrations section gives an example, describing how particular Web shortcomings can be addressed within the proposed architecture. The remaining sections describe our implementation progress, plans for further work and concluding remarks.

2 Related Works

2.1 Adapting Component Based Mobile Applications Development

Mobile applications modeling expertise requires both domain knowledge and software knowledge. Mobile Applications modeling disciplines are rapidly accumulating in terms of languages, codified expertise, reference models, and automated tools. The areas where such technologies are extensively practiced, the quality features re neither of main concern nor adequately tackled. It is a well-known truth that CBD is important for large and complex systems but why it is important for mobile device applications. It tackles vital concerns such as productivity, high level of abstraction, partitioning of the system development process from the component development process and reusability [5]. Reusability offers a number of advantages to a software development team. An assembly of component assembly leads to a 70% reduction in development cycle time; an 84% reduction in project cost, and a productivity index of 26.2, compared to an industry norm of 16.9. For the development of mass mobile examination system, CBD is a smart method, but due to its explicit requirements such as real time, safety, reliability, minimum memory and CPU consumption, standard component models cannot be used [5]. Rather than, a new CBD methodology is very much needed for the development of mobile mass examination system to deal with its specific requirements.

Mobile applications frameworks development model in this paper is based on component based software development. One of the principles of computer science field to solve a problem is divide and conquer i.e., divide the bigger problem into smaller chunks. This principle fits into component based development. The aim is to build large computer systems from small pieces called a component that has already been built instead of building complete system from scratch. Software companies have used the same concept to develop software in standardized parts/components. Software components are shipped with the libraries available with software (Fig. 1).

While compositional adaptation is restricted to the middleware and application layers, parameter tuning also applies to lower layers, such as the operating system, protocols, and the hardware [11]. For example, at the protocol level, the TCP dynamically adjusts its window to avoid or recover from network congestion. Also, at the hardware layer, adaptations could target ergonomics (e.g., adjust the display brightness), power management (e.g., turn idle network adapters off), etc. An adaptive system can be abstracted by a number of layers. The hardware layer, which includes all hardware devices, is right below the operating system and protocols layer. These layers have the common characteristic that any changes in them affect the whole device. For example, if the display brightness or the processor speed is adjusted, all applications using them are affected. Similarly, changes at the operating system layer also affect the whole device. For example, the Windows Mobile operating system allows the adjustment of the storage versus the program memory

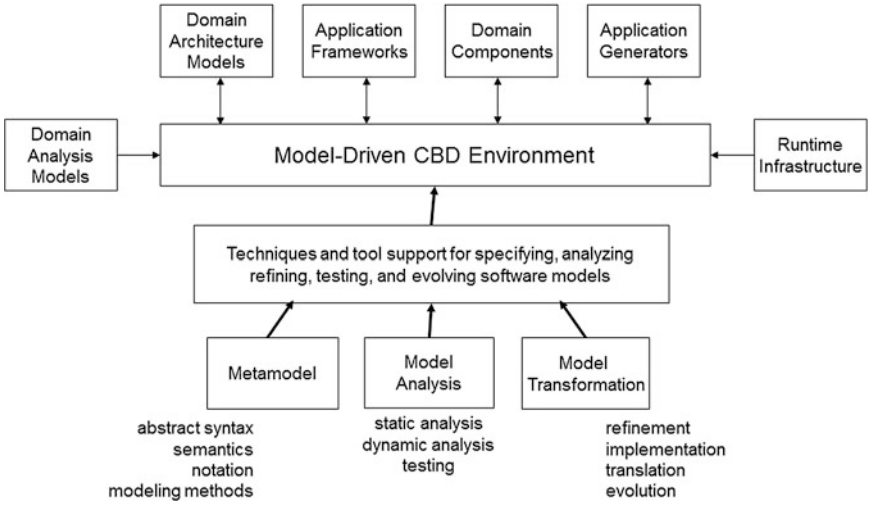


Fig. 1 CBD driven mobile applications development

balance. Clearly, any change in this balance affects the device, and consequently all applications running on it. On the other hand, changes performed at the layers of components and component-based applications have a more limited scope. For example, it is possible to replace a component implementation, or adjust one of its parameters, without causing any direct effect to the applications that are not using it. At the application layer, adaptations are typically achieved with dynamic reconfiguration (classified as changes to the software implementation, composition, or distribution [5]). Finally, besides being limited to these layers, adaptations can also extend beyond the boundaries of a single hosting device. This type of adaptations, which are quoted as distribution adaptations, is of particular interest to this paper. It is argued that users can experience great enhancements in the quality they perceive in their software services if the used devices are capable of synergistically sharing services and components, thus better utilizing the available resources [8].

2.2 CBD Process for Mobile Applications

CBD promises cost-effective productivity assuring a high flexibility and maintenance by assembling the components as independent business processing. The parts. The CBD environment is divided into two Features according to process evolution level. That is, we consider the CBD process as a supply process producing and providing the commercial components into a repository, and consume process supporting component utilization for constructing business solutions [14, 15]. The big picture represents essential works for realizing the CBD process,

subjecting the basic principles for component reuse that is acquisition–understanding-applying. CBD process looks different from a traditional one. The development of components, and the composition of an application from the components, are separated. Typically, the two process parts will be executed by different organizations, the component manufacturer and the organization that wants to license and reuse the manufactured components. We refer to these organizations as the component developer as reuse for component and the application composer as reuse with component, respectively as in Fig. 2. Component development is a traditional development process since all the usual lifecycle phases are traversed. The main difference is that the end product is not a complete application. This means that the product is comparatively small, which may make development processes suited to small projects preferable. CBD has rapidly become substantial and interesting field in business applications. Especially, since CBD is primarily used as a way to assist in controlling the complexity and risks of large-scale system development, providing an architecture-centric and reuse-centric approach at the build and deployment phases of development. So now, many vendors and researchers have tried to establish the CBD maturity by involving the following strategies [16, 17]:

(1) Efficient building of individual components,
(2) Efficient building of development solutions of in a new domain effectively,
(3) Efficient adapting a existing solutions to new problems and efficient evolution of sets of solutions.

But, by the lack of standardization and clearness for the CBD approach method, we can't expect a practice benefits in business solutions. So, we need the approach techniques in each step for organizing and practicing the CBD process like a Fig. 3 [6].

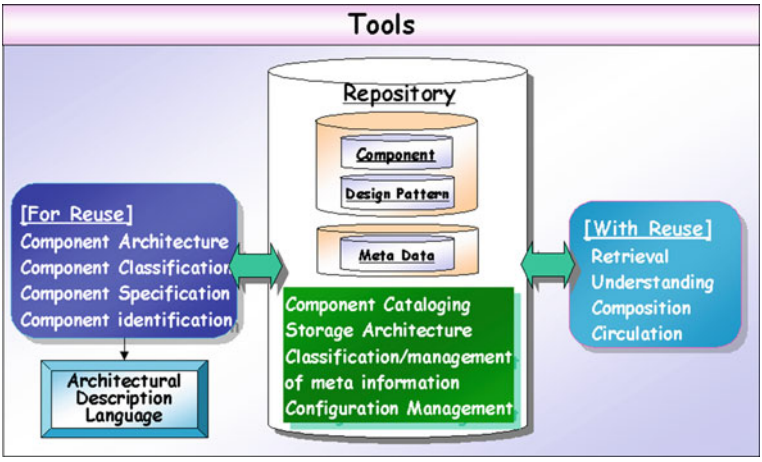


Fig. 2 Basis techniques for mobile applications development using CBD process

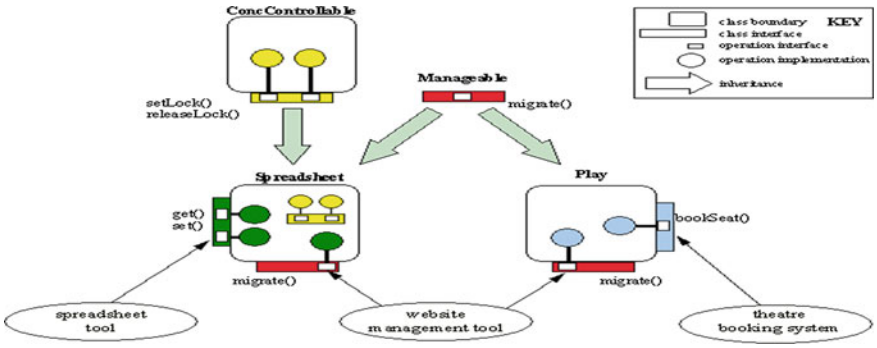


Fig. 3 Illustration of using the adapting component model with interactive mobile applications development component model

2.2.1 Component Interfaces

In an ideal situation, component interfaces would be formally specified, and a CBD would perform formal reasoning to ensure the semantic compatibility of component implementations with their interfaces. However, such reasoning tools are still not widely available or widely used by practitioners, and most commercial components do not have formally specified interfaces. A global namespace of interfaces partly solves the problem of how a CBD will ensure consistency between the semantics of a provided component and the semantics required of the component [18, 19].

While there may be different interfaces providing the same functionality in a global namespace of interfaces, two interfaces with the same name are intended to be functionally equivalent. On a fundamental level, this greatly simplifies the problem of matching provided components to required semantics, since the problem is reduced to name equality. Only when components do not match at the interface level is human intervention required: Either they are truly incompatible (i.e., incompatible on a semantic level), or the incompatibility is only syntactic, so that they can be matched by simple manual adaptation (for example by wrapping one of them). Of course, mechanisms are still needed to ensure that a component correctly implements the semantics promised by its interfaces, but this problem already existed along-side the component matching problem.

3 Design of Interactive Mobile Applications Development Using Adapting Component Model

The primary componentize of our research is to develop an extensible adapting component model with interactive mobile applications development infrastructure which is able to support a wide range of resources and services. Our model makes extensive use of the concepts of component-orientation to achieve the necessary

extensibility characteristics. Within this component-Based Development framework, proven concepts from the distributed component-Based Development community will be applied to the problems currently facing the AHMS. The interactions between the system components are described in the section entitled "System Architecture," which is followed by a section entitled "Adapting component model with interactive mobile applications development component properties" which classifies and describes a collection of properties applicable to different classes of Adapting component model with interactive mobile applications development Component.

3.1 Interactive Mobile Applications Development Using Adapting Component Model

In common with the current Web, the proposed adapting component model with interactive mobile applications development component architecture consists of three basic entity types, namely, clients, servers, and published components.

In the current Web and mobile environment, these three types correspond to Web browsers (e.g., mosaic), Web daemons (e.g., CERN HTTPD), and documentation resources (e.g., HTML documents) respectively. Our architecture supports both client-component (client-server), and inter component (peer-to-peer) communication.

Figure 3, illustrates the logical view of client-component interactions within the Adapting component model with interactive mobile applications development component architecture. A single server process is shown, managing a single Adapting component model with interactive mobile applications development component (although servers are capable of managing multiple components of different types), which is being accessed via two different clients, a standard Web browser, and a dedicated bespoke application. This diagram highlights *interoperability* as one of the key concepts of the architecture, that is, the support for component accessibility via different applications using multiple protocols. As stated earlier, Adapting component model with interactive mobile applications development components are encapsulated, meaning that they are responsible for managing their own properties (e.g., security, persistence, concurrency control etc.) rather than the application accessing the component. For example, in the case of concurrency, the component manages its own access control, based upon its internal policy, irrespective of which application method invocations originate from. The local representation of a component, together with the available operations, may vary depending upon the particular type of client accessing it (Fig. 4).

Although, it has been already stated that we believe CGI to be too low-level for direct programming, CGI interfaces to remote components can be automatically created using stub-generation tools. We have implemented a basic stub-generator, which uses an abstract definition of the remote component, and ANSA have

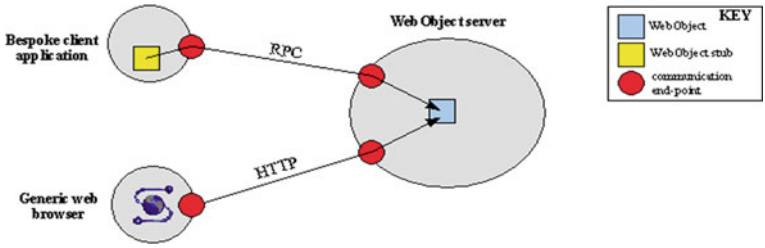


Fig. 4 Client-component interactions for adapting component model with interactive mobile applications development

recently released a more complete tool based on CORBA IDL. Recent developments using interpreted languages within the Web, including Java and SafeTcl are potentially very useful for developing client-side interfaces to Adapting component model with interactive mobile applications development Components. Using such languages, complex, architecture-neutral, front-ends dedicated to a particular Adapting component model with interactive mobile applications development component class can be developed, supporting rich sets of operations.

3.2 Components and Variation Points

A prototype component framework has been implemented, using the Java language [8]. The components are defined as classes, annotated with both required and optional metadata. Furthermore, the framework implements standardized component containers [3], providing runtime support for dynamic adaptations and life-cycle management. Similarly to the majority of industrial component models (e.g., CCM, COM and EJB), the components are considered as similar to object-oriented classes in the sense that they are instantiated and their instances can be stateful. Supporting dynamic compositional adaptations (i.e., through dynamic reconfigurations) is a major research area itself. Kramer and Magee have detected a number of important issues for dynamic reconfigurations, most notably the requirement for quiescence [8]. The component framework defines a set of metadata which are required to enable dynamic adaptations. These metadata are defined inline with the code using annotations. The attached metadata define information such as a unique identifier, a list of roles they implement and a list of roles they export.

The annotation-based mechanisms are used for specifying both the offered and the required roles of components. These roles are then used to facilitate the dynamic composition of applications. The framework achieves dynamic composition by dynamically adding and removing bindings between components, thus enabling the dynamic configuration and reconfiguration of component-based applications.

In this manner, the framework acts as a broker, managing the available and the required roles. Different composition plans are formed by matching required

services to offered ones. The actual binding of the components is achieved with the use of reflection, which is a standard feature of Java. With the use of the annotation mechanisms, the components expose both their required and their offered roles. Using these metadata, the components can be connected to each other to form a composition.

Furthermore, hierarchical composition is achieved by implementing the external view of a component through another composition. One of the main advantages of this framework is that it allows the dynamic planning of compositions, as opposed to frameworks which require a predefined set of possible adaptations. While this paper focuses on compositional adaptation, further adaptivity (e.g., at the hardware layer) is also possible (i.e., parameter tuning) [8].

3.3 Adapting Component Model with Interactive Mobile Applications Development

In order to construct adaptive applications, the developers specify how an application should be composed, and when. The first part is achieved with the construction of components with the use of roles and variation points. The latter also requires a mechanism to reason on the context and to select variations. Naturally, these two requirements separate the development phase in two parts: developing the application logic and defining the adaptive behavior. An apparent advantage of this approach is that the same components can be reused for the development of additional, adaptive applications (naturally inherited from the component-oriented approach). Furthermore, the same adaptation strategies can be reused in the context of different applications. For example, a strategy which monitors the network requirements of an application, as a function of its components, can be reused for different applications as well.

Finally, because of the high variability which characterizes mobile environments, it is important that the adaptations can be decided and implemented in a quick and efficient manner (i.e. to cope with frequent and unpredicted disconnections). The following paragraphs describe the two required phases. In addition to client-component communication, our architecture also supports inter-component communication, regardless of the components' location. In effect, the architecture may be viewed as a single distributed service, partitioned over different hosts as illustrated in Fig. 5. Inter-component communication is used for a variety of purposes, including referencing, migration, caching, and replication. In addition to Adapting component model with interactive mobile applications development Components, servers may contain Adapting component model with interactive mobile applications development component stubs, or aliases, which are named components that simply forward operation invocations to another component, transparently to clients. One particular use of aliases is in implementation of name-servers, since a name-server may be viewed simply as a collection of named

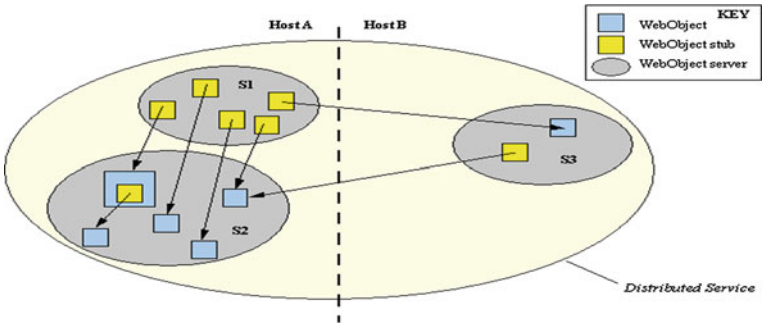


Fig. 5 Inter-component interactions

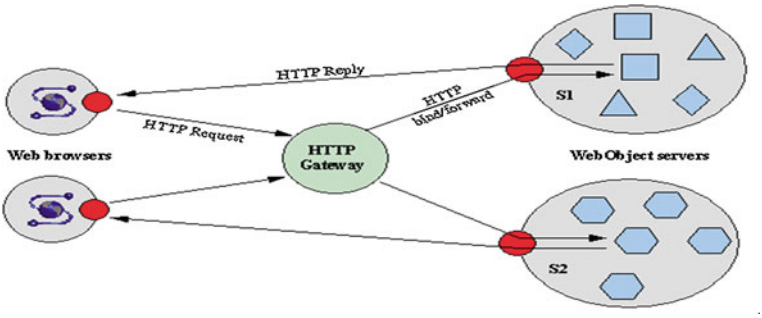


Fig. 6 Client-component communication through gateway

components which alias other components with alternative names (server S1 in diagram). Components may also contain stubs to other components (shown in S2 in diagram). This feature is used in our implementation of referencing, which is described further in the "Illustrations" section.

One method of interfacing with multiple servers is to make use of an HTTP Gateway, which uses stub components to forward component invocations through to the appropriate server. The gateway is transparent to clients accessing the components; incoming requests are simply forwarded to the destination component, which parses the request and replies accordingly. This is illustrated in Fig. 6, in which server S1 manages a number of different types of component (illustrated by different shapes) and server S2 manages components of a single type. As the processing of operations is entirely the responsibility of the individual component, the introduction of new component types is transparent to the gateway.

Based on critiques of the current mobile by ourselves and others [10], and also our experience with distributed systems in general, we have attempted to identify the set of properties that are required by Adapting component model with interactive mobile applications development Components. We have classified these properties into three categories: core properties, common properties, and

class-specific properties. In this section we shall present what we believe to be the core properties required by all Adapting component model with interactive mobile applications development Components and give examples of some common properties.

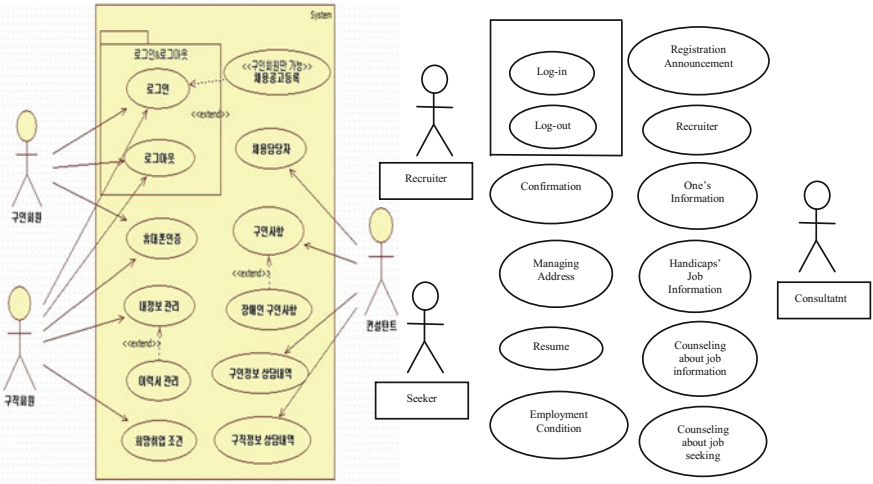
Four properties have been identified as being the core requirements for adapting component model with interactive mobile applications development components: Naming, Sharing, Mobility, and Referencing. The implementation of these properties is divided between the components themselves and the supporting infrastructure, which manages the components. Each property will be considered in turn.

Naming: One of the fundamental concepts of the component-Based Development paradigm is identity. The ability to name a component is required in order to unambiguously communicate with and about it. Context-relative naming is an essential feature of our environment so as to support interoperability and scalability. As mentioned previously, different clients may use different local representations of a remote component (URLs, client-stub components, etc.). Since it is impractical to impose new naming conventions on existing systems, we require the ability to translate names between system-boundaries. Furthermore, for extensibility, we need to be able to incorporate new naming systems. Within our design, naming is provided via the component infrastructure.

3.4 Implementation of Adapting Component Model with Interactive Mobile Applications Development

Having described our model in the previous sections, we shall now illustrate how two of the core properties, referencing and mobility, are implemented within the model. Our aim is to address the current problem of broken links and provide transparent component migration.

In our model Web resources are represented as Adapting component model with interactive mobile applications development Components and may be referenced from some *root*, either directly, via Adapting component model with interactive mobile applications development component stubs, or by being contained within another Adapting component model with interactive mobile applications development component (note that Adapting component model with interactive mobile applications development component stubs are themselves Adapting component model with interactive mobile applications development Components). This is illustrated in Fig. 7, which shows a number of components, all of which are reachable from some roots. Our service maintains the distributed referencing graph and uses reference counting to detect unreferenced components. Stubs, when created, perform an explicit *bind* operation on the component they refer to (thereby incrementing the component's reference count) and perform an *unbind* operation whenever the stub is deleted (thereby decrementing the count).



(Translated in English)

Fig. 7 Adapting component model with interactive mobile applications development modeling using UML

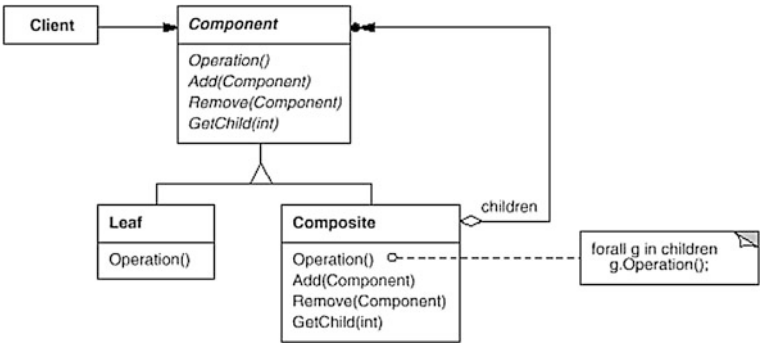


Fig. 8 Possible compositions for the schedule manager application

3.5 Dynamic Composition

At this point, the developer has specified a number of component implementations, along with a set of specifications about the roles they offer and the roles they require. Given these, the framework can plan a set of valid compositions, as illustrated by Fig. 8.

So far, in this phase the developers have defined the set of components, along with their offered and required roles. These artifacts however, cannot result to an adaptive application until the framework is instructed on how and when each composition is selected. As it has been argued already, the task of defining how the application is adapted is a different concern, which should be kept as independent as

possible from the task of defining the core application logic. In this respect, it is the responsibility of the second phase to define which composition is more suitable for each context in an independent and reusable manner. The actual evaluation of the utility functions takes place when relevant context changes occur. At that point, the framework evaluates the utility as a function of the new context and properties, and decides whether a new composition can improve on the existing one.

4 Conclusions

The Adapting component model with interactive mobile applications development component model, presented in this paper, is intended to provide a flexible and extensible way of building Web and mobile applications, where Web resources are encapsulated as components with well-defined interfaces. Components inherit desirable characteristics, redefining operations as is appropriate; users interact with these components in a uniform manner. We have identified three categories of component properties: core, common, and specific, and have described an implementation using the core properties which addresses what we believe to be one of the most significant problems facing the current mobile that of referential integrity. A key feature of our design is support for interoperability; for example, in addition to sophisticated clients which may use the rich component inter faces that our model provides, our implementation will also allow Adapting component model with interactive mobile applications development Components to continue to be accessed using existing mobile browsers.

We also describe the mobile applications frameworks and adaptive component model, which we have developed, based upon these ideas, and show, through a prototype implementation, how we have used the model to address the problems of referential integrity and transparent component (resource) migration. We will show the prototyping applications using our approaches. Indications were given on future work.

Acknowledgements This research was Supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC (Convergence Information Technology Research Center) support program (IITP-2016-H8601-16-1007) supervised by the IITP (Institute for Information & communication Technology Promotion).

This research was also supported by the International Research & Development Pro-gram of the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (Grant number: K 2014075112).

References

1. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges, IEEE Personal Communications, vol. 8, no. 4, pp. 44–49, (2001) August.

2. Czyperski, C.: Component Software: Beyond Object-Oriented Programming, ACM Press/Addison-Wesley, (2008).
3. OMG, Common Component Request Broker Architecture and Specification, OMG Document Number 91.12.1
4. Ogbuji, U.: The Past, Present and Future of Mobile Services, http://www.Mobile services. org/index.php/article/articleview/663/4/61/, (2004).
5. Barnawi, A., Qureshi, M. R. J., Khan, A. I.: A Framework for Next Generation Mobile and Wireless Networks Application Development using Hybrid Component Based Development Model", International Journal of Research and Reviews in Next Generation Networks (IJRRNGN), vol. 1, no. 2, pp. 51–58, December (2011).
6. Litoiu, M.: Migrating to Mobile Services-latency and scalability, Proceedings of Fourth International Workshop on Mobile Site Evolution, pp. 13–20, October (2002). URL: http://www.tigris.org/
7. Brown, A.: Using service-oriented architecture and component-based development to build Mobile service applications," Rational Software white paper from IBM, (2002). 4.
8. Paspallis, N., Papadopoulos, G. A.: An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns, Proceedings of the 30th Annual COMPSAC'06, (2006).
9. Soley, R. and OMG Staff Strategy Group.: Model Driven Architecture, OMG Whit Paper Draft 3.2, at URL: http://www.omg.org/~soley/mda.html, (2000).
10. Poole, J. D.: Model Driven Architecture: Vision, Standards and Emerging Technologies," European Conference on Object-Oriented Programming, at URL: http://www.omg.org/mda/mda_files/Model-Driven_Architecture.pdf, (2004). 4.
11. Qureshi, M. R. J.: Reuse and Component Based Development," in Proc. of Int. Conf. Software Engineering Research and Practice (SERP'06 Las Vegas, USA), pp. 146–150, 26-29 June (2006).
12. Champion, M., Ferris, C., Newcomer, E., Iona, Orchard, D.: Mobile Services Architecture: W3C Working Draft," http://www.w3.org/TR/ws-arch/, (2002).