Abstract. The objective of the UIF process is to aid smart phone application development. The method relies on UIF diagrams on designing application user interaction and application class structures. Earlier publications have defined processes for standalone applications and application performance assessment.

This paper extends the method to cover designs where the application is constructed using multiple independent or semi-independent modules. Constructing application in modular fashion offers multiple benefits for application development. The main benefit is that application logic can be split into smaller and easier to manage parts, which then could be developed, tested and installed to the target devices separately.

Keywords. UML, component based, modeling, smart phone.

1. Introduction

This paper focuses on designing smart phone applications UIs and applications structures in a modular approach in which the applications can be developed using independent or semi-independent component modules to operate together composing the whole functional application. This approach offers a wide range of benefits for application development.

As stated in (Moschoyiannis S., et al.. 2003) the idea of modular design is that the final system can be assembled from prefabricated software components, thus increasing the scope for reuse and replacement. Other benefit identified in (Zhou J. Shorter 2003) is that component-based architecture has potential to remove the tight coupling of application parts, thus enabling reuse and parallel development of different parts and reducing development time as well as giving to developers clear time-to-market advantage.

Components can be defined in many different ways, for example (Pernici B., et al. (2000) defines a component as a coherent package of software implementation that can be independently developed and delivered.

In (Guereis C.R et al. 2001) the component is defined as a binary piece of software, self-contained, customizable and composable, with well-defined interfaces and dependencies. The definition in (Jung M and Biersack E. 2000) then mainly combines both of these definitions. Components (Jung M and Biersack E. 2000):
- Are largely de-coupled, it means they can be independently developed and delivered.
- Have explicit and well-specified interfaces for the services they provide;
- Have explicit and well-specified interfaces for services it expects from other components, and
- Can be customized and composed with other components without modification of code.

The component concept used in this paper differs slightly from the previous definitions. Besides, when discussing internally reusable objects we refer them as components, although (Pernici B., et al. 2000) refers them as objects used in traditional object-oriented development. The reason why they do not fit the definitions in some cases is mainly due that some application parts identified as components by UIF process might not be enough independent to be delivered independently.

In this paper a modified UIF process for designing modular applications is introduced. The method is used to identify applications parts as independent or semi-independent components. Then it is used to model the applications architecture in modular fashion comprising of multiple components working together.

All changes to development steps from the original UIF process as described in (Silvennoinen j. and Avatchanakorn V. 2/2005) are discussed and a real world example application design is used to illustrate the usage and usability of the UIF process in modular application design for the changed parts of the UIF process.

2. Overview of the method

The method introduced for modular smart phone applications design is divided into following eight steps:

1. Use Case Modeling with UML Use Case diagrams;
2. Modeling the system with first level UIF diagrams;
3. Refining First level UIF with possible additional main UIs;
4. Identifying main components and support UIs for the main UIs;
2.2 First level UIF modeling

The second step with the UIF process is to derive the first level UIF diagrams from the Use case diagrams. With the modular application development this step is also used to further identify the modularization aspects for the application with following steps:

1. Identifying application main modules and their responsibilities.
2. Identifying possibilities to share components between modules.
3. Modeling first level UIF diagrams for modules and components.

With the first step the application is split to the modules according to the Use Case diagrams main use cases, or use case groups. The responsibilities for each module should also be identified, and should include all aspects defined in use cases. The second step is to identify if there would be any functionalities that could be encapsulated into independent components and shared between the application modules. As a last step the first level UIF diagrams are modeled according to the information gathered in the first two steps. Different modules and shared components should be drawn using component diagrams to identify that they are implemented separately.

Similarly as with the first level UIF diagram modeling the second level diagrams for different modules and shared components should be drawn using component diagrams to identify that they are implemented separately. Otherwise this step is executed as described in (Silvennoinen j. and Avatchanakorn V. 2/2005).

2.3 Modeling modularization aspects

In this step the application modularization aspects are identified and modeled. This step is handled by:
- Modeling application structures and their execution order, and
- Modeling application communication

The application architecture and the modularization method used for the application modules are chosen with the application structure modeling. Any restrictions on the execution order of the applications are also considered in this step.

The communication method depends on the application architecture as well as the modularization method chosen, but quite often there are still few different solution options. Also some application could use multiple software architectures and communication methods within them. The communication method modeling provides also the opportunity to identify data access problems and find solutions for them as well as describe all necessary steps for initiating the communication as well as maintaining it.

After this step application class diagrams are derived from the second level UIF diagrams similarly as explained in (Silvennoinen j. and Avatchanakorn V. 2/2005). The difference with modular application design is that instead having one application to design, there could be several separated applications to design for. Also there could be shared components that should be designed in a separate class diagram.
3. Example Application design

The example application is called N-SMS, and the requirements for it are defined as follows. The application needs to be able to:

1. Handle SMS messages from standard folders
   - Move/copy messages between folders
   - Send and schedule standards SMS messages
   - Modify existing messages
   - View existing standard SMS messages
   - Construct new standard SMS messages
2. Handle encrypted SMS messages
   - Send and schedule 3DES encrypted messages
3. Construct fake incoming SMS messages
   - Construct and schedule SMS messages that appear to be sent to the phone by selected party
   - Allow modifying messages while they are waiting to be scheduled.

It was also desired that the application would be made extensible, so future additions could be easily incorporated into the system. Also required functionalities should be made in a way that they could be inserted/removed from the application without disturbing other components.

3.1 UML Use Case modeling

The first task is to find the main use cases for the applications, and since this application is to be designed in modular fashion where different functionalities are handled in different applications, the first task is to find the tasks groups and to bind them into the main use cases. When evaluating the requirements defined for the application, it can be seen that there are three main requirements groups: (i) Managing standard SMS messages and mail folders; (ii) Managing encrypted SMS messages; and (iii) Managing fake SMS messages.

Next task in this step is to define the use cases for each main use cases shown in main use case diagram. The Figure 2 shows the use case diagram for Manage mail boxes Use Case.

3.2 First level UIF modeling

In the use case modeling activities, it was already determined that the application would be constructed with three different application modules. In architectural design decision an important issue is to determine whether the different modules would be constructed as independent applications, so they could be used individually without other applications, or as semi-independent ones, whereas one application would act as a main application, and others would be only executed by using the main application.

With the N-SMS application it was determined that using the semi-independent application approach worked better. This was due the fact that the manage mail boxes application could be used as a general message handler and as a message selection handler for the other application components that are handling messages stored in standard messaging folders. Also as mentioned in (Silvennoinen j. and Avatchanakorn V. 1/2005 and 2/2005) each UI drawn into the UIF diagram is thought to be an independent object, which can be reused in other implementations. With modular applications these independent UI components could also be reused internally between application modules. For this reason during this step it is also important to look for these components. Reusing the UI components between different application modules can
help reducing the coding work for the applications as well as it also will give more constant overall look and feel for the application.

With the N-SMS application was determined that all components needed to be able to construct and view messages, and for this reason these functionalities should be put in a separate module that could be used by different modules, resulting the first level UIF diagrams for the N-SMS application as shown in Figure 5. The notation used in the figure differs from the notations used in (Silvennoinen j. and Avatchanakorn V. 2/2005). The reason for this is that the first level UIF diagrams shown in (Silvennoinen j. and Avatchanakorn V. 2/2005) are illustrating the actual UIs used in the application and in the Figure 5, the boxes are illustrating application components, rather than their UIs. The purpose for this notation is to distinguish the different application module components and to define their responsibilities as well as the UI flows between different application modules.

As shown in use case diagram for the extra components, it has four use cases: (i) Construct message; (ii) View message; (iii) Select message; and (iv) Modify message. The message constructing and viewing are implemented in separated independent components, which leaves two UIs to be implemented in extras component. The resulting first level UIF diagram for extras component is shown in Figure 8.

The use case diagram for main application has five use cases, and similarly with other components the message composing and viewing are handled in separated components, and that leaves three UIs to be constructed for this components: Folder selection, Message selection, and Message handling. The message handling has three included use cases from which the copy/move and send/schedule are not UI related operations, from which it can be concluded that the third UI should be named as modify message UI. The resulting first level UIF diagram for the main application is shown in Figure 9.

3.3 Identify and model modularization aspects

The N-SMS application is build from semiindependent components and the modularization method for the application was selected to be document embedding. The reason behind this choice is that it allows applications to be run in modal...
fashion and in the contexts of one thread. The communication was implemented using two interfaces: MSOneNSMS and MSOneNSMSEmbedInterface. The MSOneNSMS is defined as:

```cpp
class MSOneNSMS
{
public:
virtual MSOneNSMSEmbedInterface* GetHandler()=0;
};
```

MSOneNSMS interface is implemented by the document class of the embedded application and the purpose for this interface is that the embedding application can request the interface for UIs by using GetHandler function. MSOneNSMSEmbedInterface then implements the functions for executing the UIs and is defined as:

```cpp
class MSOneNSMSEmbedInterface
{
public:
virtual void StartHandlerL(TInt aId)=0;
virtual void StartComposerL(TInt aId, CMsvSession* aS, CClientMtmRegistry* aReg)=0;
virtual void StartViewerL(TInt aVId, CMsvSession* aS, CClientMtmRegistry* aReg, CArrayFixFlat<TMsvId>* aArr, TInt aCurr)=0;
};
```

Figure 10 shows a sequence diagram that illustrates how the embedded applications are run. As it can be seen from the figure, embedding application constructs the document class of the application, which then also constructs the application user interface class for the embedded application. Then the embedding application asks for the MSOneNSMSEmbedInterface by using the MSOneNSMS interface implemented in the document class. As it can be seen from the Figure 10, with this example application the MSOneNSMSEmbedInterface is implemented in application user interface class. After retrieving the MSOneNSMSEmbedInterface interface, embedding application calls the appropriate function to construct the required UI for the selected operation.

With the earlier steps in the UIF diagram it was already defined that the applications would be implemented in three different parts. In this stage the application structure should also be identified for each application package, so all necessary modules and their basic functionalities could be identified accordingly. With the UIF process for modular applications it is suggested to use deployment diagrams to identify different installation packages as well as the communications issues inside the packages.

The main application deployment diagram is shown in Figure 11. The only components needed for this application are the main application itself and an optional amount of DLL modules. The amount of DLL modules depends on the implementation. Basically any non-UI related mail folder management code could be put into a static DLL that could be shared between different application modules. Also the UI codes for composer and viewer modules could be put into a DLL to reduce the installation package size with embedded applications.

The deployment diagram for Encrypted SMS application is shown in figure 12. This application implements a new type of bio message, and for this reason it is required to implement the components needed for identifying and handling the incoming bio messages. This functionality is handled by bio parser component, which also requires a BIF file to be present in the system.

The Fake SMS application is required to schedule incoming Fake SMS messages even while none of the applications are running. This indicates a need for a
background application which would be used to achieve this functionality. Fake SMS messages should also be sent according to their scheduling time, whenever the phone is turned on, even when the application has not been started by user. This indicates that the application needs to be started automatically when turning on the phone. Fake SMS messages are constructed with the Fake SMS application and sent with the background sending application, which indicates that there should be a shared store to store the messages. The deployment diagram for the Fake SMS application was constructed according to reasoning above and is shown in Figure 13.

With the design the SMS messages are stored in standard RDbNamedDatabase that is accessed only by one application at any time. The background SMS sending application is identified in the figure 13 as a server application.

4. Conclusions

As shown in this paper the UIF process can also be used on designing modular component based applications architectures. As it was also illustrated with the partial smart phone application design, the UIF process based approach offers simple, yet powerful method on deriving the components of the system as well as designing the whole interaction needed for the applications architectural design.

The drawbacks of the UIF process with modular application design is the same as identified in the earlier publications, the method requires experienced domain expert to conduct most of the activities included in it.

Currently UIF process includes only steps for deriving the applications conceptual class diagrams, as well as partial performance assessment. With future research the performance assessment should be extended to cover all performance aspects as well as the UIF process should also be extended to cover implementation issues.

6. References


