Identification of Design Patterns for Mobile Services with J2ME

J. Narsoo and N. Mohamudally
School of Business Informatics and Software Engineering, University of Technology, Mauritius, La Tour Koenig, Pte Aux Sables, Mauritius

jnarsoo@utm.intnet.mu alimohamudally@utm.intnet.mu

Abstract

The functional approach to software development has been used for years and new methodologies have evolved, including, object-oriented approach. Objects are sharing an important part in the software business. They have become the building blocks of many systems whether commercial applications, real-time applications or embedded systems applications. As objects represent an abstraction of the real world, systems modelled using objects can be easily read and interpreted. Design patterns have been used on desktop systems. Today, the trend is towards building applications on small hand-held devices like mobile phones, PDAs or pocket PCs. Due to the limitations of hand-held devices, applications should be optimised and design pattern is the right choice to cater for this need.

In this paper, the design pattern chosen for mobile services using J2ME is the One-function structure design patterns. Although it resembles the behavioural design pattern, the One-function structure name has been given because the pattern is made up of other classes and objects and is performing a single task. The patterns have been used in two applications with basic management functions namely add, edit, erase and search and the result obtained proves their re-usability and reliability.

Keywords: Software Development, design patterns, one-function structure design pattern

Introduction

Patterns are used and repeated in everyday life. For, example, to make a cup of tea, we need to boil water, add tea, milk and finally sugar. The steps could be executed in any order but will the preparation process remain efficient, optimise and quality be the same at the end? Design Patterns is the result of the work of Christopher Alexander, an architect who in the early 1960's started to see urban planning and architecture as the continual re-application of some basic principles of best practices. According to Christopher Alexander “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

Software design process is very similar to other design processes in real life as these processes are executed according to certain patterns and requirements. We are always worried about how we can produce something very quickly and according to specific requirements. Software engineering has evolved from functional-oriented approach to object-oriented approach. Design patterns make use of object-oriented approach to solve design problems.

What is more interesting in the object-oriented approach is that, the different models produced reflect an abstraction of the real life situation. We only talk about classes and objects, their collaborations or interactions. If we can imagine, a software system is like an embedded system consisting of components (patterns). In an embedded system, whenever a particular module is faulty, we just replace it by another module or a particular module can be used in another system. The same idea can be applied in software system. We can create a common pattern that can be used by other systems.

According to the Gang of Four (GoF) (Gamma, Helm, Johnson, & Vlissides, 1995), "A design pattern systematically names, motivates and explains a general design that addresses a recurring design problem in object-oriented systems." It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of object and classes that solve the problem. The solution is customised and implemented to solve the problem in a particular context.

Whenever a mathematics problem needs to be solved, the problem must be read and understood. Then by using appropriate values and formulae, equations can be formulated. These equations can be solved simultaneously to get the solution. The same rules can be applied to solve other related mathematics problems. If the problem solving part can be thought of as a black box or object, then by feeding values to the black box the solution can be obtained. In this way, the black box can be used to solve similar problems. The idea behind this analogy is that the problem solving part or the black box is similar to a pattern. A pattern is a set of rules to address a particular problem and it can be designed and implemented using an object oriented approach. However, in order to have an optimised pattern, design pattern is the solution. No matter how complex a problem can be, there will always be a solution that will address the problem. How efficient the solution will be is another issue to be taken into consideration.

A general definition of design pattern could be a set of rules or steps to accomplish a task and this pattern can be used over and over again in other applications. For example, the same query pattern can be used in a library system, stock control system, airline reservation system among others.

There has been an important consideration for design patterns in the modelling of systems. France and Ghosh propose a UML-based pattern specification technique. The pattern specification technique can be used as a base for tools that support creation and evolution of patterns, and rigorous application of design patterns to UML models.

Mak, Choy, and Lun (2004) in their workon, a precise modelling of design patterns in UML, suggest the structural properties of design patterns which describe the abstract nature of pattern structures. Their objective is providing a modelling language that describes precisely the abstract nature of design patterns.

Guéhéneuc, Sahroui, and Zaidi (2004) make an experimental study of micro-architectures, which form part of a program architecture and which are similar to design patterns. They find that classes in the design patterns share the same fingerprints (quantitative signatures). They use these signatures to improve the design pattern identification algorithms.

Floyd, in his book (2002), describes a set of EJB patterns. He also suggests how the knowledge of these patterns can be applied in a use-case driven manner.

Design Patterns for Mobile Services

Design patterns have been identified to be efficient and useful in large scale and embedded systems. Mobile services are heterogeneous, large scale and embedded. The services range from chat, SMS, WAP application, M-commerce application among others. These services are not based on a standard format. One service may be very similar to other services not related to mobile and yet it may follow its own design. Standard design patterns could help to make mobile services more efficient, useful and "timeless" to implement or to be used over and over again.

Mobile phones and other hand-held devices have limited resources in terms of storage (primary and secondary) and processing capability. A poorly designed application, even if small, may take up all the resources and eventually slow down or block the device. When designing applications for small devices great care should be taken in order to have the right application running according to requirements and as far as possible consuming only a small amount of the limited resources. Examples of design patterns in larger applications have been demonstrated with Java 2 Enterprise Edition (J2EE) and Enterprise Java Bean (EJB). On the other hand, Java 2 Micro Edition (J2ME) is the development platform for hand-held devices and embedded systems. With J2ME, optimised services or applications can be implemented by making use of design patterns. In this paper, Java 2 Microedition has been used to demonstrate mobile services.

J2ME (Java 2 Microedition) is a compressed version of the Java API and Java Virtual Machine that is designed to operate on devices with limited resources like embedded computers and microcomputers. J2ME is meant for developers of small computing devices which need to support cross-platform functionality in their products. Examples of devices that could incorporate J2ME applications are mobile phones, pagers, personal digital assistants, set-top boxes, and vehicle telematics systems. The J2ME architecture consists of a variety of configurations, profiles, and optional packages. Each combination is optimized for the memory, processing power, and input/output capabilities of a related category of devices. The result is a common Java platform that takes full advantage of each type of device to deliver a rich user experience.

To support J2ME a device must have a minimum of a 96x54 pixel display that can handle bit-mapped graphics. At least 128 kilobytes (KB) of non-volatile memory is required to run Mobile Information Device (MID) and 8KB of non-volatile memory for persistent application data.

A J2ME application consists of a main midlet which acts as the execution environment for the functions in the application. In this paper, the record management store (rms) has been used as database for the test application.

Pattern Selection

Which of the existing twenty-three design patterns could be more appropriate for designing applications or services on small devices bearing in mind of their limitations? To justify the choice of the right design patterns, two small applications have been used as test cases and these applications are the phone book and reminder systems which perform basic management functions, namely: add, edit, erase and search. J2ME development platform has been used and testing has been performed with the use of a J2ME emulator.

The phone book application allows the user to add basic details about a particular person or company like the phone number and name. The user is also able to edit and modify existing phone details. Search and delete facilities are also provided. The reminder application consists of similar services as the phone book. The user can add basic reminder details like date, time and message. Edit, search and delete services are also provided. The designs of these two applications could be performed using the following approaches.

Case 1 – Integrated Behaviours Structure

Figure 1 shows the phoneBook class which consists of its data members and all behaviours required to manipulate the class. Figure 2 shows the reminder class which consists of its data members and all behaviours required to manipulate the class.

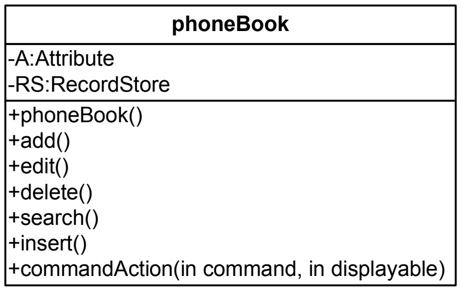

Figure 1: phoneBook Class

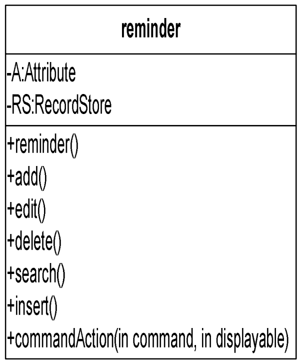

Figure 2: reminder Class

The phoneBook class is responsible for adding, modifying, searching and deleting records from the phone book record store.

The reminder class is responsible for adding, modifying, searching and deleting records from the reminder record store.

In both classes, the behaviours are integrated, that is, they are encapsulated within their classes. It can be observed that both classes are offering similar functions and yet they have been implemented independently in separate classes. The implication of applying this approach is the requirement constraint in terms of primary and secondary memories. The bigger the classes, the higher will be the memory usage.

Case 2 – One-function Structure

Instead of having all the behaviours integrated in one class and having them duplicated for other classes as shown in Figures 1 and 2, each method can be implemented through a separate class. In this way, a class will perform exactly one function. Figures 3, 4, 5 and 6 show extracts of the One-function structure design patterns. The basic idea behind this approach is to have the behaviour acting as a pattern and use it over and over again in the same application or other applications. In other words, the pattern is re-usable.

Of course these patterns should be designed in such a way so that they can accept parameters and be customised automatically at run time according to the applications or systems they are involved.

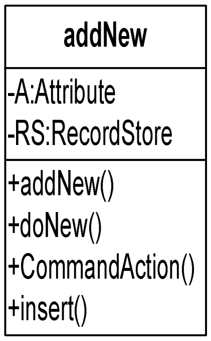

Figure 3: AddNew Class
The Add class is responsible for adding a new record in the record store.

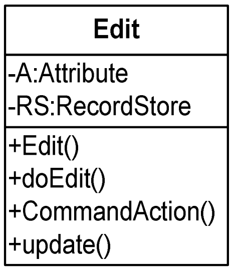

Figure 4: Edit Class
The Edit class is responsible for modifying existing records in the record store.

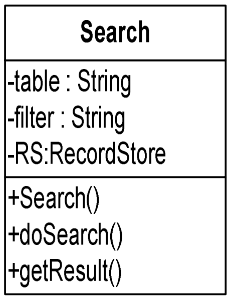

Figure 5: Search Class
The Search class is responsible for querying the record store according to a filter.

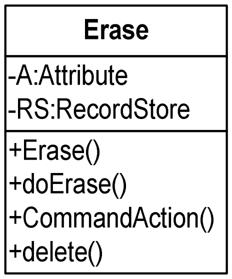

Figure 6: Erase Class
The Erase class is responsible for deleting records from the record store.

Integrated Behaviours Structure versus One-function Structure

Table 1: Comparison of Integrated Behaviours and One-function structures

Memory Usage	**Integrated Behaviours Structure**		**One-function Structure**	
	Phone Book	**Reminder**	**Phone Book**	**Reminder**
Run-time	5 Kb Approximately	5 Kb Approximately	2 Kb/function	2 Kb/function

Both applications using the integrated behaviours class design are offering identical services and yet each one is implementing its behaviours separately, thus increasing the size of the class and eventually the storage and memory usage at run-time. From table 1, it can be observed that the One-function structure design is better compared to the integrated behaviours structure design since the memory usage at run-time is lower. One-function structure applications will load only one function at a time in the memory, thus taking less storage and executing faster.

The One-function structure design pattern is similar to the behavioural design pattern. The design pattern has been named One-function structure as it is composed of other classes and objects and it is performing only a single task.

When implementing the four basic management functions within a J2ME application, we may have the impression that the total size of the application, at run-time, includes the sum of individual size of each four patterns. This is not the case, because whenever a service is required then, at that particular point in time, the pattern or function is instantiated (object created). Thus, the size of the application is much less.

One-function Structure Design Patterns

Four basic management functions have been chosen to illustrate the One-function structure design patterns. These are the add, edit, search and erase patterns. A detailed description of each of these patterns is given below.

addNew Pattern

Intent

Allows the addition of a new record.

Motivation

Adding records in tables in a database system can make use of a similar function although the fields may be different. Sometimes, it is necessary to have a structure performing a single function so that it can be used in many different systems.

The addNew pattern can be used in any J2ME applications where the add function is re-

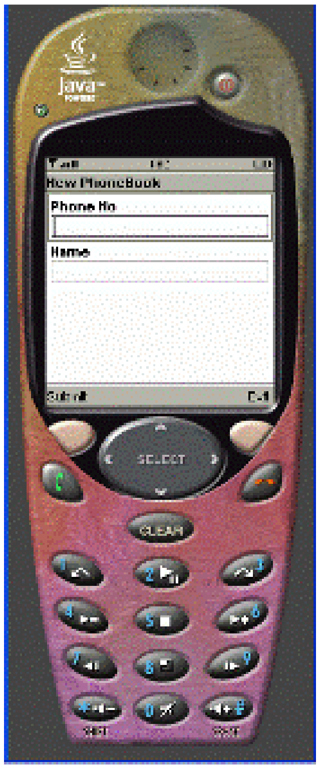

Figure 7: Add phonebook

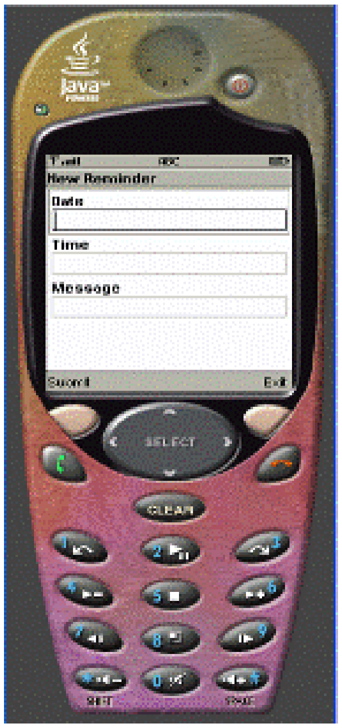

Figure 8: Add reminder

quired. It has been discussed that the addNew pattern can be used to add new details in the phonebook and reminder applications.

According the system, the addNew pattern displays a form with the appropriate items, allows the user to enter his/her details and saves them.

Figure 7 shows the add pattern used in phonebook, in execution using an emulator. Figure 8 shows the add pattern used in reminder, in execution using an emulator.

Applicability

The addNew pattern can be used or customised for any J2ME system provided that appropriate parameters are passed to the pattern before execution. (See Figure 9.)

Structure

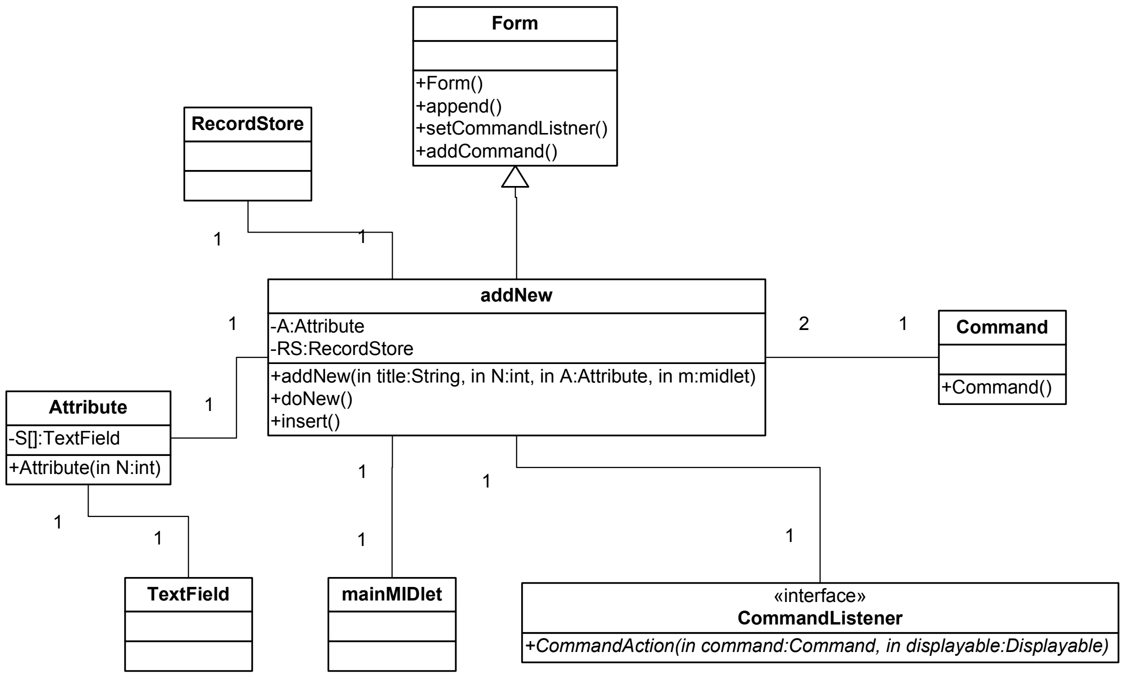

Figure 9: addNew design pattern

Participants

Form (Node)

This is used to create the class AddNew through inheritance.

addNew

This defines the constructor AddNew and doNew and insert operations.

Command

This declares an interface for executing an operation.

RecordStore

This is used to store data that has been entered by the user.

Attribute

This defines a collection of data items which will be displayed on the user interface

CommandListener

This defines an interface that captures an operation executed by the application.

TextField

This is used to display a single text box.

Collaborations

Figure 10 shows the interactions, that is, message exchanges between the different objects involved in the add operation.

- The addNew object is created and the constructor addNew() is executed to initialize the object and displays a blank form. The title field represents the function of the structure together with the name of the application. Two command buttons are created, one to submit the data entered and the other to exit the form.
- The doNew method is then executed to append the textfields, passed as parameters, to the pattern.
- The two command buttons are added to the form and the setCommandListener() method is executed.
- The user can then enter data in the textfields.
- When the CommandAction() method is executed, the insert() method is also started.
- The RecordStore object is opened for writing. A new record is inserted by executing the method addRecord().
- The RecordStore object is the closed by executing the method CloseRecord-Store().

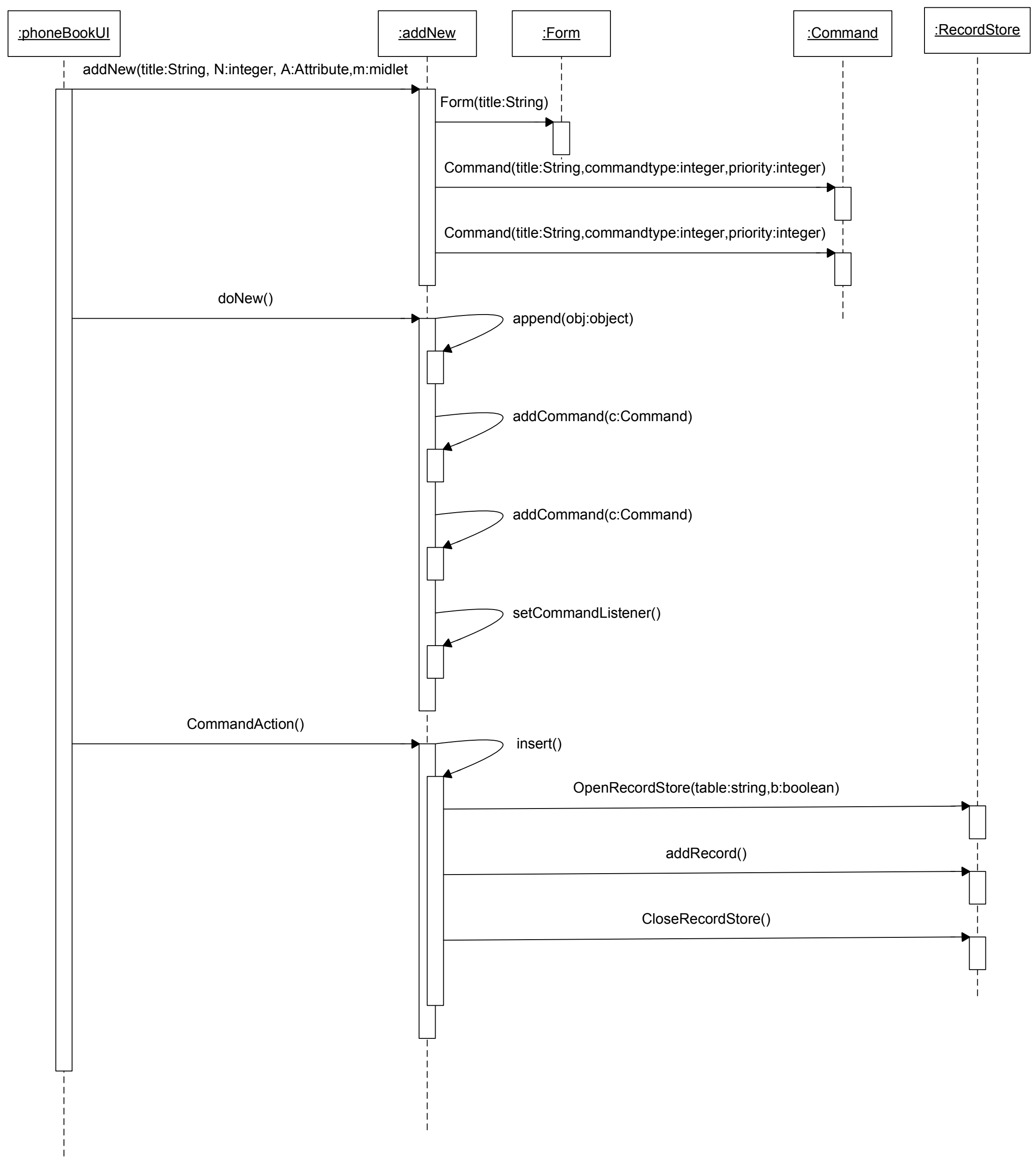
:phoneBookUI
:addNew
:Form
:Command
:RecordStore
addNew(title:String, N:integer, A:Attribute,m:midlet
Form(title:String)
Command(title:String,commandtype:integer,priority:integer)
Command(title:String,commandtype:integer,priority:integer)
doNew()
append(obj:object)
addCommand(c:Command)
addCommand(c:Command)
setCommandListener()
CommandAction()
insert()
OpenRecordStore(table:string,b:boolean)
addRecord()
CloseRecordStore()

Figure 10: add sequence diagram

Implementation

Appropriate parameters must be supplied to the class addNew constructor. These are the title of the option, the attribute list, the number of attributes and the main midlet.

Edit Pattern

Intent

Allows the modification of an existing record.

Motivation

Modifying records in tables in a database system can make use of a similar function although the fields may be different. Sometimes it is necessary to have a structure performing a single function so that it can be used in many different systems.

The edit pattern can be used in any J2ME applications where the edit function is required. It has been discussed that the edit pattern can be used to modify existing records from the phonebook and reminder applications.

According the system, the edit pattern displays a form with the appropriate retrieved items, allows the user to enter his/her details and saves them.

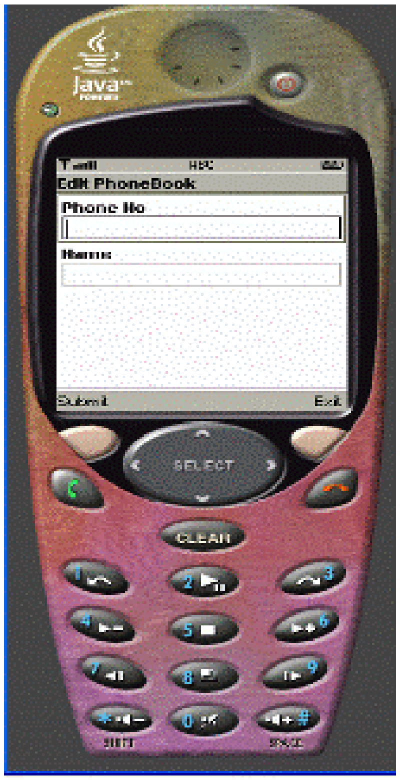

Figure 11: Edit phonebook

Figure 11 shows the edit pattern used in phonebook, in execution using an emulator.

Applicability

The edit pattern can be used or customised for any J2ME system provided that appropriate parameters are passed to the pattern before execution.

Structure

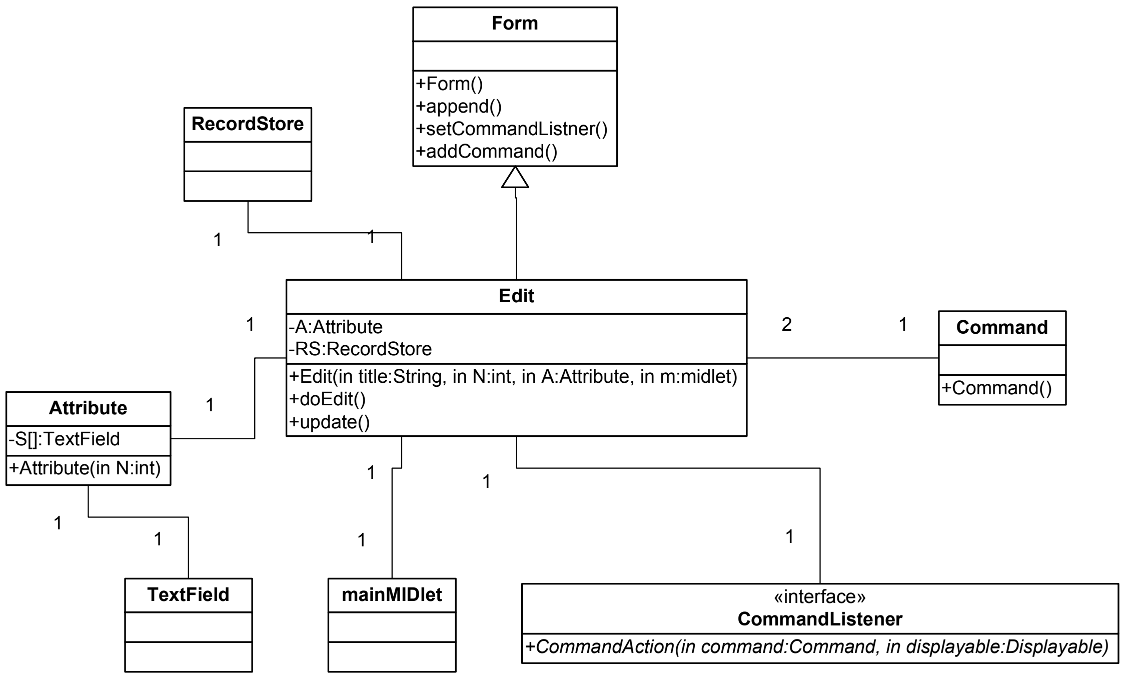

Figure 12: Edit pattern

Participants

Form (Node)

This is used to create the class Edit through inheritance.

Edit

This defines the constructor Edit and doEdit and update operations.

Command

This declares an interface for executing an operation.

RecordStore

This is used to store data that has been entered by the user.

Attribute

This defines a collection of data items which will be displayed on the user interface.

CommandListener

This defines an interface that captures an operation executed by the application.

TextField

This is used to display a single text box.

Collaborations

The following diagram shows the interactions, that is, message exchanges between the different objects involved in the add operation.

- The Edit object is created and the constructor Edit() is executed to initialize the object and displays a blank form. The title field represents the function of the structure together with the name of the application. Two command buttons are created, one to submit the data entered and the other to exit the form.
- The doEdit method is then executed to append the textfields with data retrieved from the record store, passed as parameters, to the pattern.
- The two command buttons are added to the form and the setCommandListener() method is executed.
- The user can then modify the data in the textfields.
- When the CommandAction() method is executed, the update() method is also started.
- The RecordStore object is opened for updating. The selected record is modified by executing the method setRecord().
- The RecordStore object is the closed by executing the method CloseRecordStore().

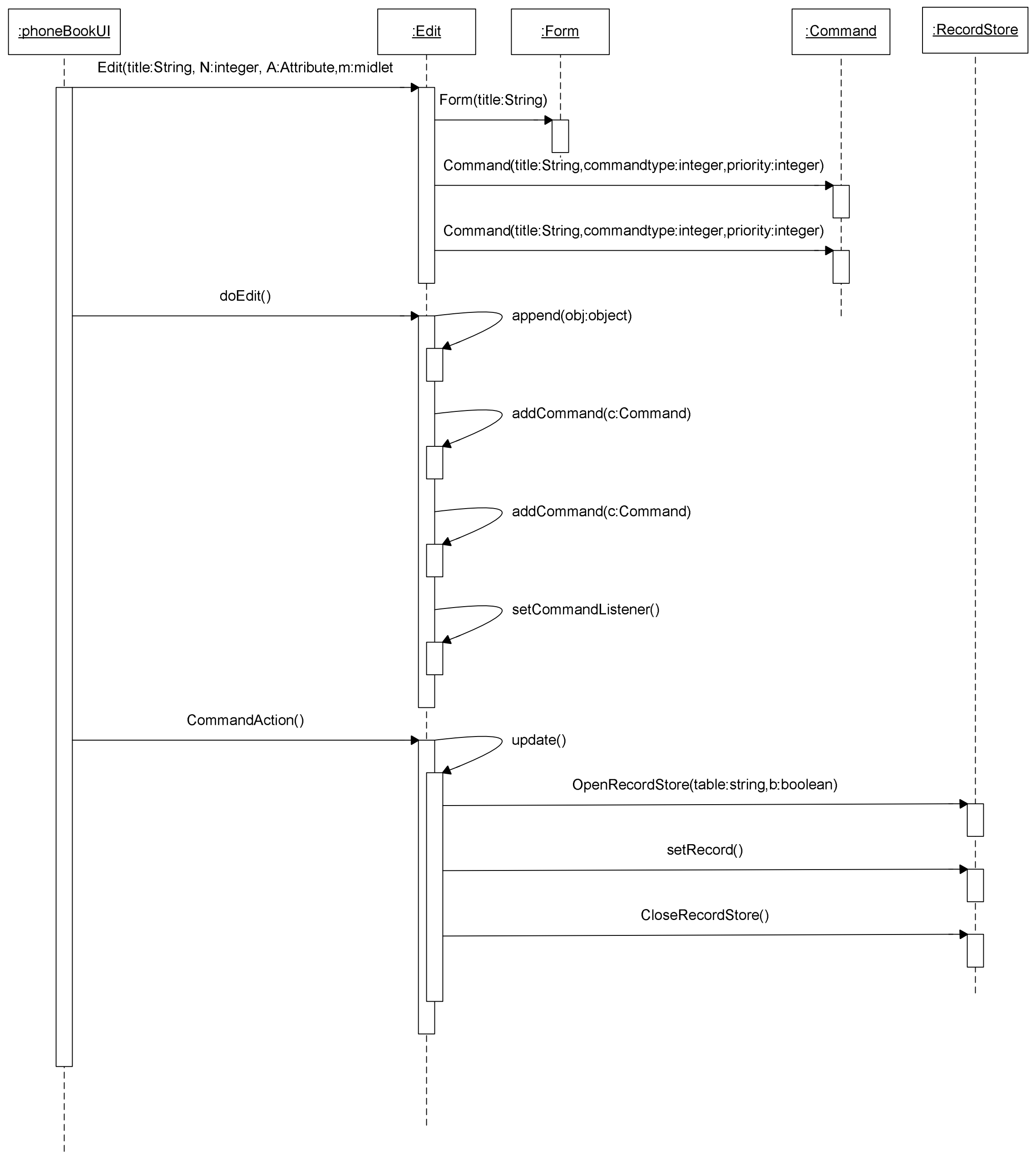
:phoneBookUI
:Edit
:Form
:Command
:RecordStore
Edit(title:String, N:integer, A:Attribute,m:midlet
Form(title:String)
Command(title:String,commandtype:integer,priority:integer)
Command(title:String,commandtype:integer,priority:integer)
doEdit()
append(obj:object)
addCommand(c:Command)
addCommand(c:Command)
setCommandListener()
CommandAction()
update()
OpenRecordStore(table:string,b:boolean)
setRecord()
CloseRecordStore()

Figure 13: Edit sequence diagram

Implementation

Appropriate parameters must be supplied to the class Edit constructor. These are the title of the option, the attribute list, the number of attributes and the main midlet.

Search Pattern

Intent

Allows for the querying of existing records.

Motivation

Making use of the search pattern will be an advantage as it helps to search record in many different systems. It is necessary to have a structure performing a single function so that it can be used over and over again.

The search pattern can be used in any J2ME applications where the search service is required.

According the system, the search pattern displays a blank form with the appropriate fields, allows the user to enter his/her details and retrieve the records.

Applicability

The search pattern can be used or customised for any J2ME system provided that appropriate parameters are passed to the pattern before execution.

Structure

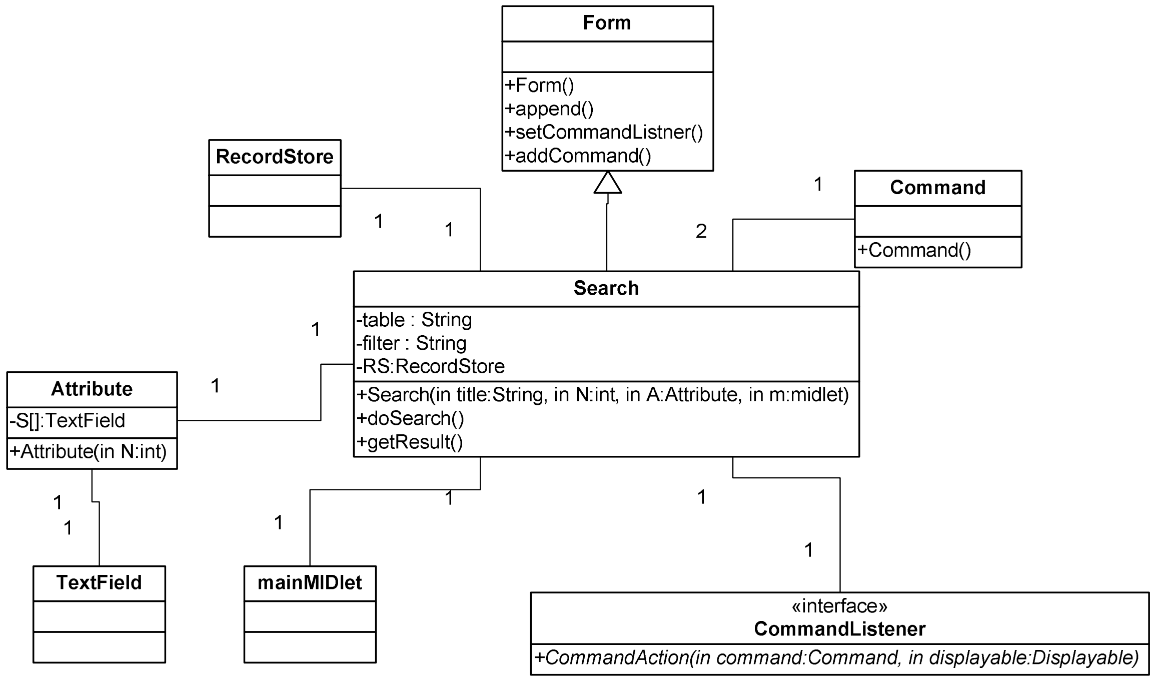

Figure 14: Search pattern

Participants

Form (Node)

This is used to create the class Search through inheritance

Search

This defines the constructor Search and doSearch operations

Command

This declares an interface for executing an operation.

RecordStore

This contains stored data.

Attribute

This defines a collection of data items which will be displayed on the user interface.

CommandListener

This defines an interface that captures an operation executed by the application.

TextField

This is used to display a single text box.

Collaborations

The following diagram shows the interactions, that is, message exchanges between the different objects involved in the add operation.

- The Search object is created and the constructor Search() is executed to initialize the object and displays a blank form. The title field represents the function of the structure together with the name of the application. Two command buttons are created, one to submit the data entered and the other to exit the form.
- The doSearch method is then executed to append the textfields.
- The two command buttons are added to the form and the setCommandListener() method is executed.
- The user can then enter the search criteria.
- When the CommandAction() method is executed, the getResult() method is also started.
- The RecordStore object is opened for reading. The selected record is retrieved by executing the method getRecord().
- The RecordStore object is the closed by executing the method CloseRecordStore().

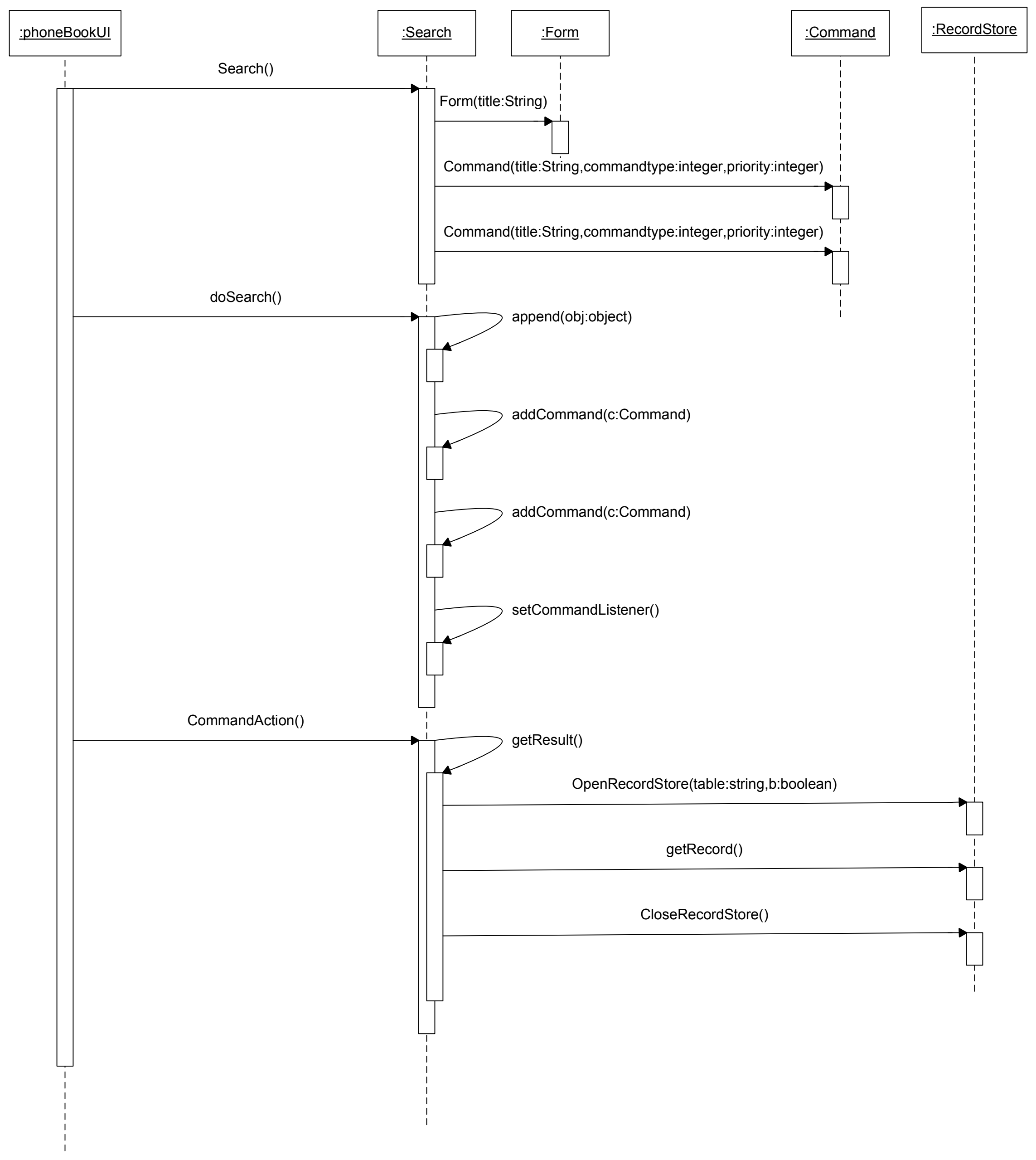

Figure 15: Search sequence diagram

Implementation

Appropriate parameters must be supplied to the class Search constructor. These are the title of the option and the main midlet.

Erase Pattern

Intent

Allows the deletion of existing records.

Motivation

Making use of the erase pattern will be an advantage as it helps to delete records in many different systems. It is necessary to have a structure performing a single function so that it can be used over and over again.

The erase pattern can be used in any J2ME applications where the erase service is required.

According the system, the erase pattern displays a form with the appropriate data.

Applicability

The erase pattern can be used or customised for any J2ME system provided that appropriate parameters are passed to the pattern before execution.

Structure

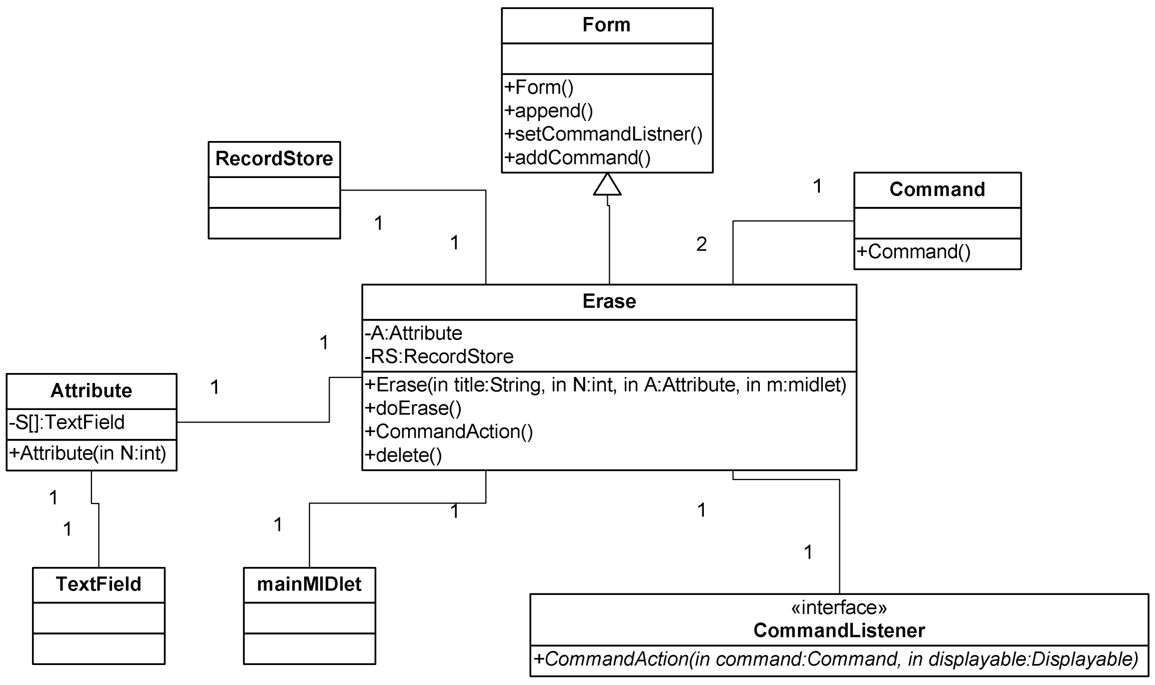

Figure 16: Erase pattern

Participants

Form (Node)

This is used to create the class Erase through inheritance

Erase

This defines the constructor Erase and doErase and delete operations

Command

This declares an interface for executing an operation.

RecordStore

This contains stored data.

Attribute

This defines a collection of data items which will be displayed on the user interface.

CommandListener

This defines an interface that captures an operation executed by the application.

TextField

This is used to display a single text box.

Collaborations

The following diagram shows the interactions, that is, message exchanges between the different objects involved in the add operation.

- The Erase object is created and the constructor Erase() is executed to initialize the object and displays a blank form. The title field represents the function of the structure together with the name of the application. Two command buttons are created, one to submit the data entered and the other to exit the form.
- The doErase method is then executed to append the textfields.
- The two command buttons are added to the form and the setCommandListener() method is executed.
- When the CommandAction() method is executed, the delete() method is also started.
- The RecordStore object is opened. The selected record is deleted by executing the method deleteRecord().
- The RecordStore object is the closed by executing the method CloseRecordStore().

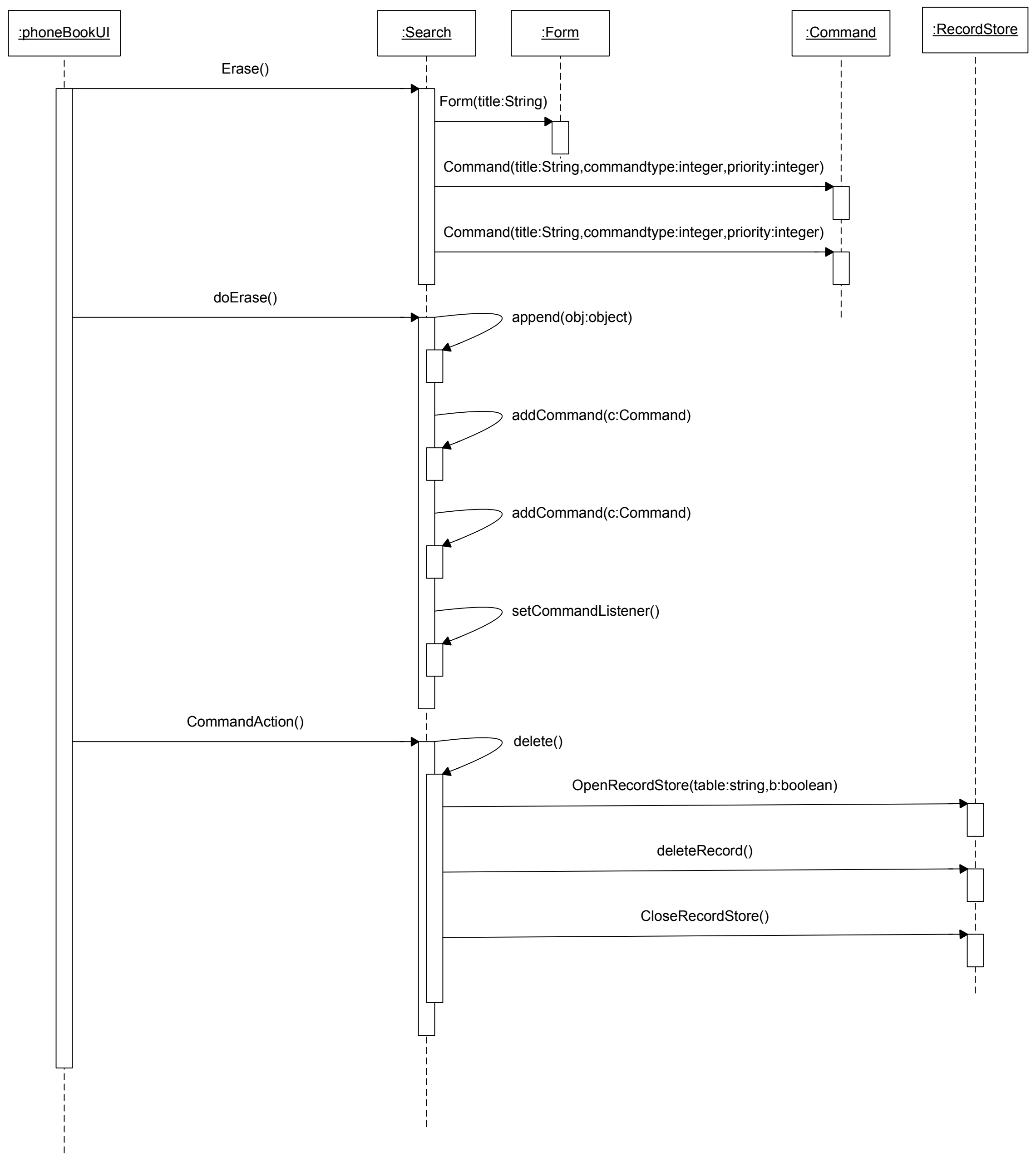

Figure 17: Erase sequence diagram

Implementation

Appropriate parameters must be supplied to the class Erase constructor. These are the title of the option, the attribute list, the number of attributes and the main midlet.

Add and Edit Patterns

It can be observed that the add and edit patterns are similar. Only one pattern could have been created for both functions and customised it with the use of an extra parameter. If we want to implement the validation rules, then their functions would be completely different.

Conclusion

The goal of this paper has been the identification of design patterns for mobile services with J2ME. Four management functions, add, edit, erase and search, have been chosen and the One-function structure design pattern has been derived for these functions.

Although the One-function structure design pattern is similar to the behavioural design pattern, it has been named One-function structure design pattern as it is composed of other classes and objects and it is performing only a single task. To be able to use the pattern on different applications, appropriate parameters are accepted. The One-function structure design pattern is a relatively small structure due to its single task activity.

In this paper, a comparison of the One-function structure design pattern with an integrated behaviour (normal approach) has been given. It has been noted that the One-function design pattern makes use of less storage and less memory at run-time. In other words, it executes faster. The reusability of the pattern has been demonstrated in two applications, phonebook and reminder. For example, the same add, edit, erase and search patterns have been used in both applications.

On the other hand, J2ME has its own limitation and many Java features cannot be implemented compared to desktop applications. We can upgrade the hand-held devices to be able to support these advanced features. But to what extent and is it really necessary? The user interface is another issue on small devices with limited screen size and input capabilities. Additional packages can be added to J2ME to improve the user interface features but this will require additional resources. In J2ME, all services provided by a particular application evolve around a single midlet. When one service is executed, a new display is available but when the service is closed, the application does not automatically return to the previous display compared to window desktop applications. The previous display can only be restored by calling the main midlet again. This is another drawback of J2ME.

The design patterns, identified so far in this thesis, represent only a small part of design patterns for mobile services. As a future work the identified patterns can be tested and implemented on various hand-held devices and have a correct performance measurement. These patterns can also be used and tested with other J2ME applications. More services could be identified and new patterns could be derived and documented and the existing services like chat, SMS among others, on hand-held devices, could be re-engineered using design patterns.

References

Canfora, G., Di Santo, G., & Zimeo, E. (2004). *Toward seamless migration of Java AWT-based applications to personal wireless devices.* IEEE Conference on Reverse Engineering.

Floyd, M. (2002). *EJB design patterns: Advanced patterns, processes, and idioms.* Wiley.

France, R. B., & Ghosh, S.(2003). A UML-based pattern specification technique. *IEEE Transactions on Software Engineering, 30*(3).

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Guéhéneuc, Y.G, Sahroui, H., & Zaidi, F. (2004). *Fingerprinting design patterns.* IEEE Conference on Reverse Engineering.

Heineman, G. T, & Councill, W. T. (2001). *COMPONENT-BASED software engineering: Putting the pieces together.* Addison-Wesley.

Keogh, J. (2003). *J2ME: The complete reference*. Mc Graw Hill.

Kim, S. K., & Carrington, D.(2004). *Using integrated metamodeling to define OO design patterns with Object-Z and UML.* IEEE Asia-Pacific Software Engineering Conference.

Mak, J. K., Choy, C. S. T., & Lun, D. P. K. (2004). *Precise modeling of design patterns in UML*. IEEE Conference on Software Engineering.

Biographies

Mr. **J Narsoo** is graduated from the University of Mauritius and University of Technology, Mauritius in the field of computer science. He is presently lecturer in the School of Business Informatics and Software Engineering at the University of Technology, Mauritius. His topics of research interests comprise object oriented software engineering, design patterns and data compression.

Dr. Nawaz Mohamudaly is graduated from the University of Science and Technology of Lille I (France) in the field of telecommunications. He is presently heading the School of Business Informatics and Software Engineering at the University of Technology, Mauritius. His topics of research interests comprise Mobile Computing, Business Process Outsourcing with emphasis on Global IT Management, Web and Educational technologies. Dr Mohamudally is currently working on a pilot project aiming at making software development a viable economic activity in the African continent. He has co-organised several international conferences namely, the International Conference in Business Process Outsourcing and Modelling, and the African Outsourcing and Contact Centre Conference. Dr. Mohamudally has established very good working relationship and collaborates with the private industry for its HR development.