Journal of Computing and Information Technology - CIT 17, 2009, 2, 123–140
doi:10.2498/cit.1000964


# mFrame: An Application Framework for Mobile Resource-constrained Computing Environments

Hinko Vincar[1], Hossein Saiedian[2] and Serhiy Morozov[2]

[1]United Nations, Cologne, Germany
[2]University of Kansas, Lawrence, USA

The component and architectural design reuse value of application frameworks enables higher productivity, faster turn-around time, and reduced cost when compared with the traditional development approaches. Because of the constrained nature of mobile devices, the application frameworks for such a computing environment have to provide a simple and lightweight implementation while still maintaining their flexibility and reusability value. This paper presents **m**obile **Frame**work (mFrame) application framework that can be used for deploying software applications to compact mobile devices such as cellular phones, personal digital assistants (PDA), global positioning systems (GPS) etc. It introduces a queuing and service layer architecture that provides simple data exchange mechanism between the presentation components and local and remote business services. Compact mobile devices using this architecture will be able to handle network disconnections, because the requests will be saved; and information will be exchanged with the remote services upon reconnection.

*Keywords:* mFrame, application framework, mobile computing, resource-constrained

## 1. Mobile Resource-constrained Devices

### 1.1. Introduction

The market for compact mobile devices like cellular phones, personal digital assistants (PDA), and global positioning systems is growing rapidly. With the advent of third generation mobile networks (3G), and other improvements in hardware design, these devices are becoming more suitable as a platform for application development. However, because of inconsistent design guidelines and lack of agreed upon specification, reusability and modifiability quality attributes are usually not considered during development of mobile applications.

When applied to mobile resource-constrained devices, application frameworks can simplify the application development. An application framework is a semi-finished application where core parts are already implemented and are in place. An application developer who uses the application framework provides the other parts and completes the application. The component and architectural design reuse value of application frameworks can enable higher productivity, rapid development and reduction in overall development cost.

### 1.2. Problem Definition

Development environment for compact mobile devices has tight memory constraints and requires very careful and controlled allocation and de-allocation of memory. The reliability of mobile networks is very diverse and unpredictable. The networks are characterized by high latency, low bandwidth and interrupted connections. Since information can be delayed or lost over wireless links, mobile applications must be able to tolerate some delays that can have a major impact on user's perception of an application's performance (Varshney 2002). An application framework for such a resource-constrained environment has to enable simple and lightweight implementation, while still maintaining its flexibility and reusability value.

This paper presents the design, implementation and usage of mFrame (**m**obile **Frame**work) application framework for deploying software applications to compact mobile devices such as cellular phones, personal digital assistants (PDA), global positioning systems (GPS) etc. The proposed application framework introduces a queuing and service layer architecture that provides simple data exchange mechanism between the presentation components and local and remote business services. Compact mobile devices using this architecture will be able to handle network disconnections, because the requests will be saved; and information will be exchanged with the remote services upon reconnection.

Multiple related research efforts focus on maintaining a reliable connection and data transfer during mobile handoffs. In fact, the majority of published material in this area is concerned with improving the shortcomings introduced by the wireless links at lower OSI layers. One of the earliest successful solutions to this problem is discussed in "Improving reliable transport and handoff performance in cellular wireless networks". The authors of this article were able to improve the throughput performance by 20 times and decrease the handoff latency by 10 times (Balakrishnan, Seshan, & Katz 1995). Similarly, Ludwig, Konrad, and Joseph focused on the link layer performance. The authors were able to achieve a 25% improvement in network throughput that was capable of resuming data exchange after minutes, hours and even days of link unavailability (Ludwig, Konrad, & Joseph 1999). A similar disconnection solution could also be done at a data level. Bayou is an infrastructure designed specifically for collaborative applications. It is capable of maintaining its database integrity across multiple users that are loosely connected at any given time. Bayou is especially well suited for mobile users, whose network connections are often slow and unreliable (Edwards et al. 1997). An alternative data level framework, called IBM Fluid Middleware, supports building applications that are not concerned with network connectivity, replication of data or concurrency. Unlike the client-server Bayou architecture, this framework adopts a peer-to-peer approach. Peers are allowed to communicate in real time as long as connection exists and with small latency otherwise. IBM Fluid Middleware is also based on a Model-View-Controller pattern with special focus on conflict resolution and synchronization issues (Bourges-Waldegg, Duponchel, Graf & Moser 2005).

A great number of existing solutions with demonstrated results, while very impressive, do not guarantee that a mobile application built on top of such transport protocols is able to gracefully handle the loss of connectivity. This research is focused specifically on this issue. When the connection is absent, the application should be aware of it, yet remain functional. All user actions are captured and transmitted into the network as soon as a connection becomes available. The mFrame architecture reduces the user's dependency on the connection availability. The main contribution of this paper is not to replace the existing reliable data exchange solutions but to implement them at a different level. Application level implementations are abstracted by other network levels and should be much easier to implement. Additionally, mFrame is perfectly suited for working in parallel with lower level solutions, thus further improving the reliability of mobile communication.

## 2. Related Works

Following is a brief description of application frameworks that address problems similar to those addressed by the proposed mFrame framework.

### 2.1. Struts

Struts is a server-side application framework based on the Model-View-Controller (MVC) architectural framework. It consists of helper classes used for the development of Java 2 Enterprise Edition (J2EE) applications based on the Model 2 design pattern (Carnell & Linwood 2003). Web applications developed using Struts framework are configured with an external configuration file. Struts includes a notion of tag libraries that helps to reduce the amount of Java code in the presentation component. Another benefit of this framework is its built-in validation mechanism. Furthermore, Struts has a support for internationalization and database connection pooling.

The Struts application framework is built around an ActionMapping structure (Carnell & Linwood 2003). This structure is used to convert HTTP requests submitted by users into application actions. ActionMapping encapsulates information about a handler that is intended to process the request.

### 2.2. Spring

Spring Framework is an open source application framework for Java. It is mostly used for implementing complex business logic applications that are very difficult to build from the ground up. It is often considered a collection of smaller sub-frameworks. Each one of these component frameworks is usually responsible for a common feature. Some of the components that are most relevant to this research include data access, model-view-controller, and remote access framework. The data access framework is usually combined with transaction management, fault-recovery and caching components, which results in a very feature-rich yet reliable solution. Similar to Struts, Spring is a request-based framework. It defines strict interfaces for every exposed service. One of the most important aspects of this framework is the simplicity and clarity of purpose for each of these interfaces (Spring Application Framework).

### 2.3. Maverick

Maverick is yet another MVC framework used to build web applications. This framework is available in Java and Microsoft .NET environments. Maverick is a minimalist framework which focuses solely on MVC logic, allowing developers to generate user interface components using a variety of templating and transformation technologies (Maverick Application Framework). In this process, Maverick uses the pipe-and-filter approach to chain the transformations. These transformations are pluggable and customizable.

Maverick application framework automatically picks from different views based on user language, browser type, or any other characteristic of the request. This framework includes a support for internationalization.

### 2.4. jShrunk

jShrunk is an open source framework for mobile application development. This framework is based on the J2ME CLDC specification and as such it is used to develop applications for cellular phones and PDAs. jShrunk includes communication libraries to help developers build mobile applications quickly and efficiently. The data structures used to transmit data between remote and local components are very compact to comply with limited bandwidth available (jShrunk Application Framework).

One of the problems we have faced while investigating jShrunk application framework is that its documentation and binaries are not currently available. The documentation we found seems to be obsolete and inaccurate.

### 2.5. Aranea

Aranea is a lightweight Java component framework. Like every other approach, it attempts to reduce the development complexity by reusing existing components. One of the main features of this approach is that each component's interface is restricted to only 4-6 methods. Such strict interaction constraints ensure low coupling and improve framework's agility. Aranea is also an event-driven framework. Each component within a network is constantly listening for user events and is able to react to them. This greatly simplifies the development process as certain functionality may be linked to user interface items without any additional code. Additionally, each component in this framework is a Java object which adds the benefit of object-oriented programming. This aspect of Aranea is especially useful for application modifiability as each component may be extended, replaced, or partially modified without any changes made to the framework code. Finally, this framework comes with pre-built components responsible for user interface, validation and request handling which make it very easy to get started on development (Mürk & Kabanov 2006).

### 2.6. Cocoon

Cocoon is a similar framework, maintained by the Apache Software Foundation. It is based on the concepts of separation of concerns and component-based development. This approach encourages using components as building blocks, putting them together to solve the problem at hand without any additional programming. Even though this framework focuses on XML and XSLT publishing, its caching mechanism is very interesting. Cocoon framework utilizes the lazy evaluation approach. In other words, the verification of the cache validity of a component is delayed until the last possible moment in the hope that it might not be necessary. In fact, the framework provides as many as 10 locations where the application may choose to use cached data. One of the main contributions of the Cocoon developers is the adaptive cache algorithm that performs cache cost calculations. This feature compares the cost of generating/transforming new data to the cost of validating the cache. The authors of this framework claim better performance than traditional caching mechanisms because Cocoon-based applications always choose the least costly option (Cocoon Application Framework).

### 2.7. Grails

Grails is an open source web application framework based on Groovy programming language (extension of Java platform). Similarly to other frameworks, Grails attempts to reduce the development time by removing the configuration aspect of application development. It encourages the "coding by convention" approach. In fact, this framework does not have an XML configuration file that other frameworks rely on. The authors of Grails claim that maintaining a complex and frequently changing configuration file uses valuable development time that could be spent elsewhere. Instead, this framework relies on naming and coding conventions to generate configuration. Additionally, this framework dynamically adds certain methods to many of its base classes. It determines their association from the type of class i.e. its name. This eliminates the need for developers to write wrapper and connectivity code (Grails Application Framework).

## 3. Design of the mFrame Framework

The proposed mFrame framework is a customized version of the Model-View-Controller (MVC) architectural framework. As the name suggests, MVC framework, shown in Figure 1, consists of a number of components that can be grouped into three main categories: model, view and controller. Model encapsulates the core business logic and maintains the state of application; View presents the results of operations; and Controller provides navigation management mechanism. MVC separates view and model components by establishing a subscribe/notify relationship between them, where view ensures that its appearance reflects the state of the model (Gamma, Helm, Johnson, & Vlissides 1997). A set of well-defined ob-

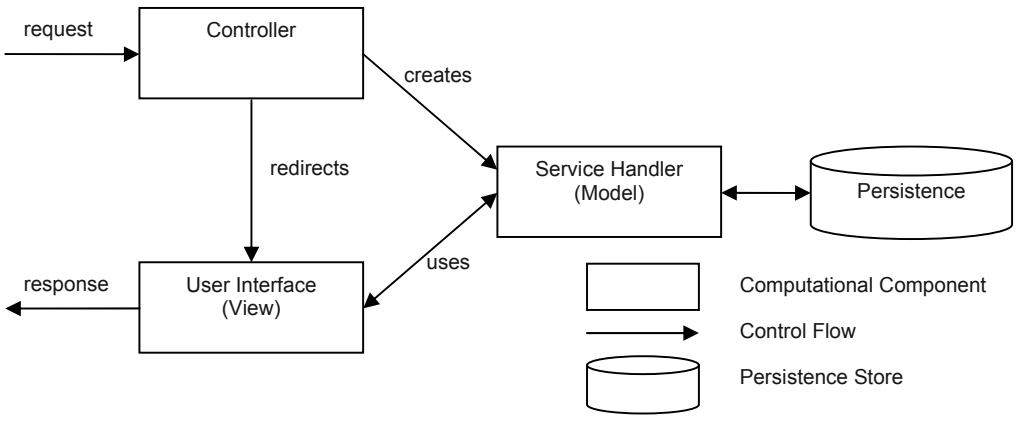



*Figure 1.* Model-View-Controller architectural framework.

jects called domain objects are used to exchange information between the components.

mFrame customizes the MVC framework to include the following:

- Service component, which includes local and innovative remote service handler implementations.
- Presentation component responsible for rendering information retrieved from the service component.
- Controller, which provides the access authentication and reconfigurable presentation and service management.
- Domain objects used to exchange information between the framework components.

Portability of the system is achieved using J2ME. J2ME is a portable, compact programming model which has been widely accepted in the mobile computing community. Using configurations (horizontal specification) and profiles (vertical specification), a rich set of APIs is provided to allow access to functions that are specific to a mobile device, without affecting portability.

Some of the most distinguishable aspects of this framework are its architecture, configuration, component coupling and cache functionality. Similarly to Struts, mFrame is using external xml configuration files. This method has been chosen as opposed to "coding by convention" approach, implemented in Grails, because the complexity and variability of the architecture does not justify loosing the configuration file. Additionally, Struts, Aranea and mFrame have built-in user input validation functionality, which reduces the required traffic. Furthermore, like jShrunk, the proposed framework provides remote data exchange abstraction in order to maintain application functionality, especially if the available bandwidth is limited. Similarly to Spring and Aranea solutions, mFrame uses few, well-defined and strict interfaces. This ensures the robustness and modifiability of this framework. Like Aranea, mFrame is event-driven, as it reacts to user actions as well as changes in connectivity. Like in many other classic MVC implementations (e.g. Maveric), the data, business logic and presentation are completely separate and handled by dedicated components. Unlike the Cocoon framework, mFrame does not offer a very sophisticated cache and cache evaluation functionality. However, there is nothing in the proposed framework's architecture that prevents it from being extended to include or mimic this feature. mFrame is, in fact, very similar to many other frameworks. The goal of this research was to create a new solution built on the success of the existing frameworks by combining successful feature implementations.

## 3.1. Top-down Architectural Model

As mentioned previously, the mFrame framework is a customized version of the MVC architectural framework. The top-down architectural model has been shown in Figure 2.

### 3.1.1. Presentation Component

The user initiates an action by submitting a request. The presentation component associated with the action creates the corresponding request and performs a basic validation. In case of error, the request is reverted back to the presentation component; otherwise, it is passed to the controller. Responsibilities of the presentation component are as follows:

- Serve as a template for concrete view implementation.
- Define command behavior of the screen components.
- Intercept the request and perform a basic user input validation.
- Decide the actions that should be performed in response to specific user events.
- Present the results of the operation.

### 3.1.2. Controller Component

Controller handles the flow of individual requests and provides access control. The requests passed through the controller are authorized against action id-based permission structure. If the permission is denied, the business transaction is stopped and the user is notified about the exception. The Controller component delegates all the requests invoked by the mobile application to the service components, and

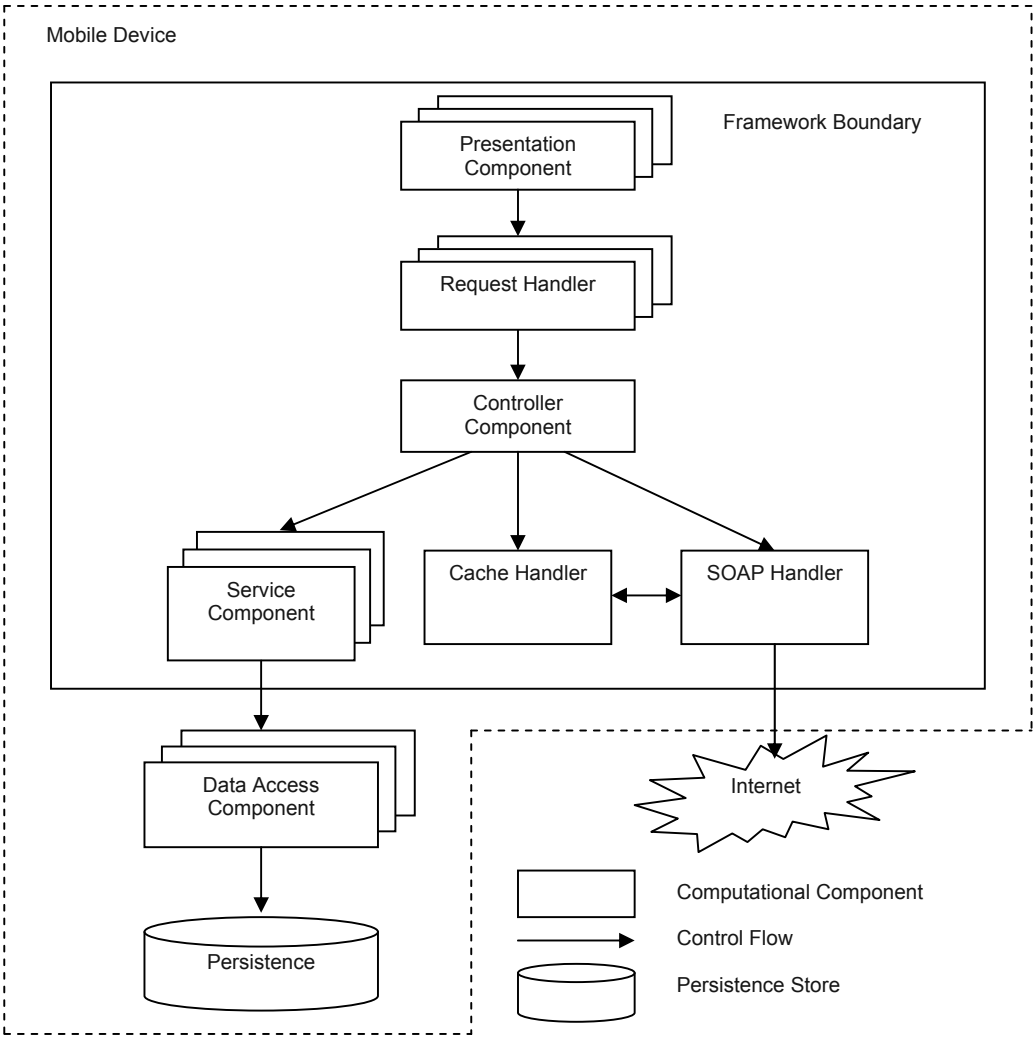



*Figure 2.* Top-down architectural model.

the responses to the presentation components. The service components and presentation components are initialized at run-time using action identifiers. In this manner, support for new business logic and its presentation can be added easily by providing new presentation and service component classes and registering them against corresponding actions. The registration is done using an external configuration file as explained in section 5.1.1. Thus the reference model dynamically adapts to the modifications done in the navigation paths.

The controller manages the following:

- Provides navigation management of the application i.e. dispatches requests to the service components and responses to the presentation components.
- Manages exceptions raised during the dispatching process.
- Handles the flow of requests and provides access control by means of action level authorization.

### 3.1.3. Service Component

The actual business logic is executed either locally or remotely. These details are encapsulated by the service component.

**Local Business Logic.** In case of the local business implementation, a service component directly invokes local Data Access Components and executes business logic. A Data Access Component provides access to mobile device's persistence layer. Since a persistence implementation is device-specific, the domain of the framework does not include this component. It

is left up to the application developer to implement this component using appropriate device-specific APIs. The device access package will include classes to execute query procedures, retrieve result sets from the storage etc.

**Remote Business Logic**. The reference architecture employs a Simple Object Access Protocol (SOAP)-based interface to exchange data with remote services via HTTP communication protocol. Support for tightly coupled synchronous remote object protocols like IIOP or DCOM is not provided, but nothing in the architecture precludes the addition of another protocol. This framework uses Enhydra kSOAP, an Open Source package designed to enable SOAP applications to run on resource-constrained environments, specifically on J2ME CLDC-based devices.

Due to the strict constraints on available bandwidth and reliability, Cache component, a communication component that is not dependent upon continuous network connectivity has been used. It utilizes the asynchronous point-to-point messaging pattern as shown in Figure 3. Users can access remote services even if the connectivity is interrupted since the information would be exchanged upon the reconnection (Bellavista, Corradi & Stefanelli 2001). Depending on the type of request, the controller component either invokes a local service repository handler or, if the connection to remote server is available, a SOAP handler. If the connection is not available, the request is queued in an Output Queue in the cache handler for future submission when the connection becomes available. The cache handler checks for the availability of external web service; and once the service is available, it processes the queued requests on first-in-first-out (FIFO) basis, using an encapsulated SOAP handler. The responses received from the remote service are stored in the Input Queue. When the request and response objects are no longer needed, the cache handler removes them safely and cleans up the resources. A request's expiration time is defined in the configuration file as explained in section 5.1.1. Thus, the service component is responsible for the following:

- Interpret input request objects
- Interact with a local persistent storage
- Control the remote execution sequence required to fulfill the request
- Pass the processing results back to presentation component

## 3.2. Design Patterns

The following design patterns have been used when developing the top-down architectural model explained above.

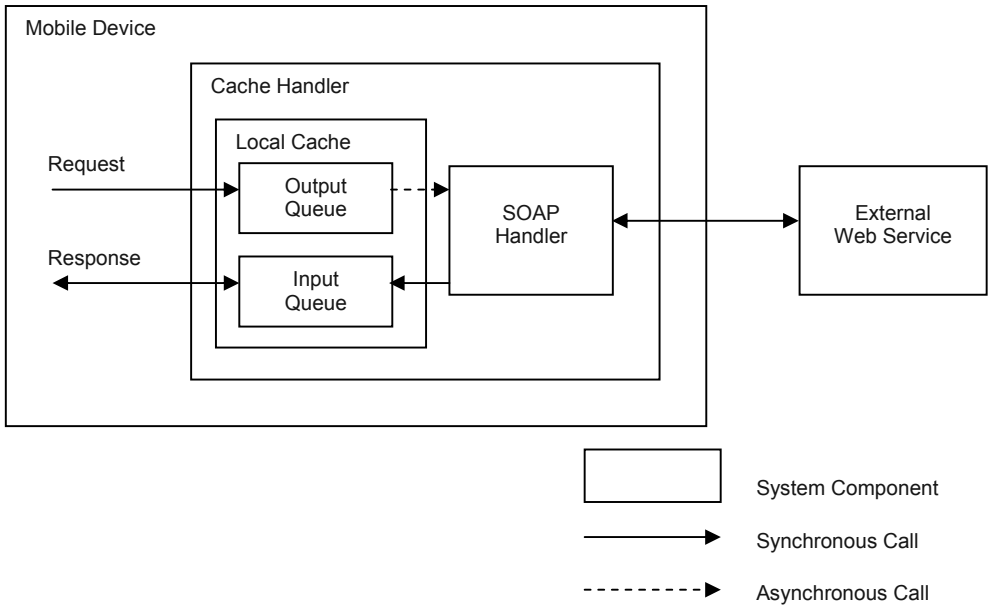



*Figure 3.* Cache component architectural model.

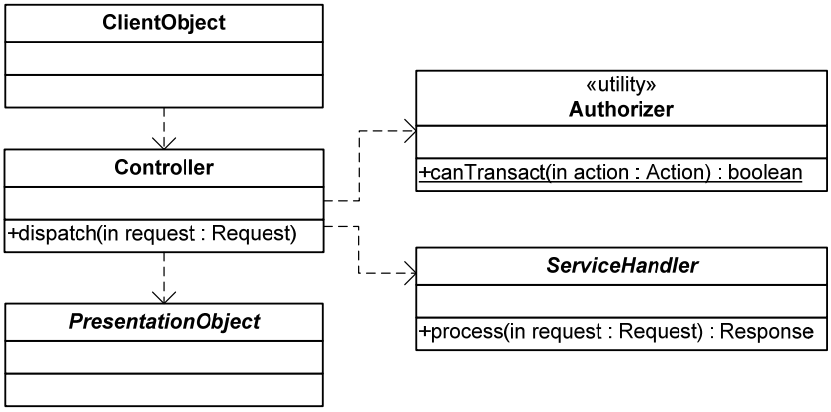



*Figure 4.* Front controller pattern class diagram.

### 3.2.1. Front Controller Pattern

The Front Controller Pattern (Alur, Crupi & Malks 2001) is used to handle the flow of information between presentation and service components. See Figure 4. This model authorizes user requests and handles exceptions that could be raised in the dispatching process. By centralizing the control, this pattern helps reduce size of the application, which is a key requirement for resource-constrained computing environment.

ClientObject submits requests to the Controller. The requests are validated against the framework's security model (Authorizer). Depending on whether the request is for local service or remote service, the Controller invokes an appropriate business object (ServiceHandler). The Controller then synchronously waits for the response from ServiceHandler. Finally, the results of processing are dispatched to the PresentationObject.

### 3.2.2. Simple Factory Pattern

Extensibility of the architecture has been achieved using the Simple Factory Pattern. See Figure 5. This pattern describes an object that creates one or more class instances in response to a certain action. All the classes that the factory object instantiates have a common super class and common methods, but the implementation of the methods is usually different.

Consider an example of a presentation object factory. The getInstance() method of PresentationObjectFactory returns an appropriate in-

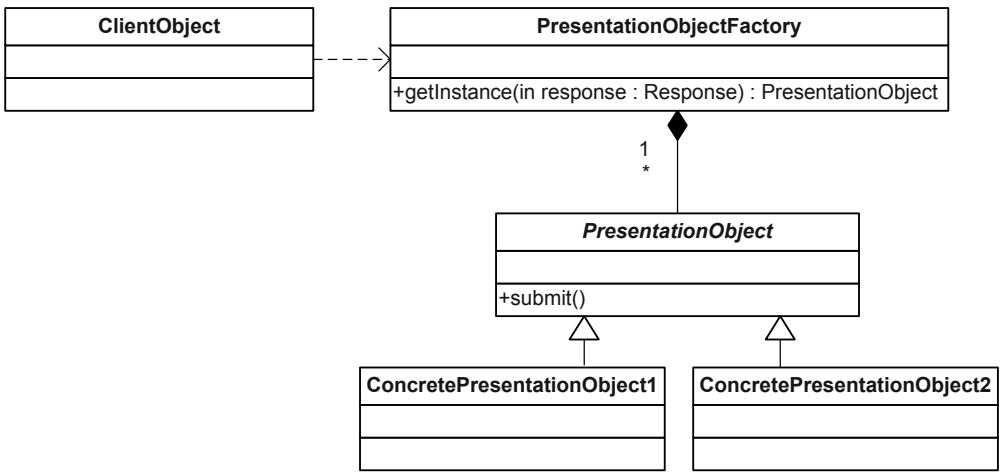



*Figure 5.* Simple factory pattern class.

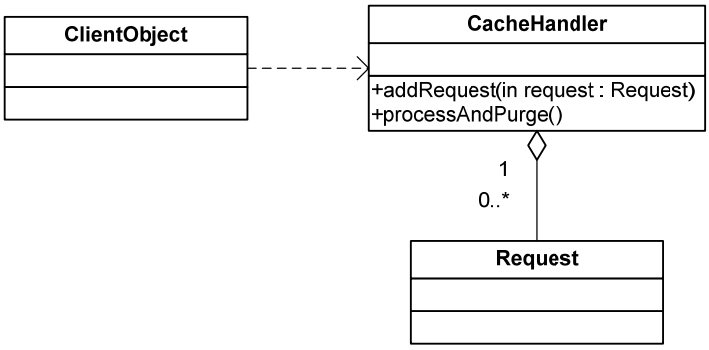



*Figure 6.* Cache management pattern class diagram.

stance of ConcretePresentationObject depending on the Response. If a new Response type is to be added, we just have to implement one more ConcretePresentationObject that extends the PresentationObject, and registers it in the PresentationObjectFactory. This pattern has also been used for instantiating a ServiceHandler, depending on the Request type.

### 3.2.3. Cache Management Pattern

Reliability of the system has been achieved using the Cache Management pattern (Larman 2004). See Figure 6. The pattern has been used to queue the requests until they can be passed to the remote service for further processing. As shown in Figure 6, the Requests for remote services are queued in the CacheHandler. The addRequest() and processAndPurge() methods are used for cache inserts and purges.

## 4. Implementation

As explained in the previous section, the top-down architectural model consists of three major components; model, view and controller. The implementation details of these components are explained in this section. Exchange of information between these components is accomplished using domain objects.

### 4.1. Domain Objects

Action object is used to store the information about the action performed by the user. It holds the class name of the service handler to be used for processing the request. In addition, it encapsulates the callback details – the presentation objects to be called in case of successful or erroneous processing (see Figure 7).

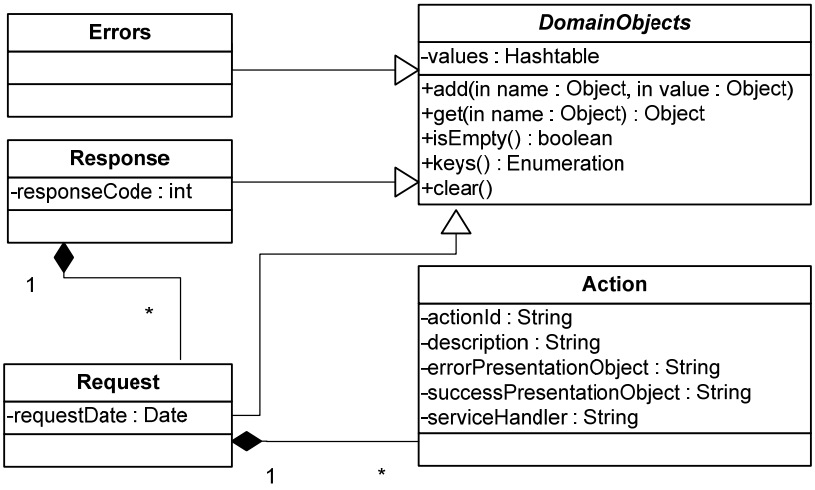



*Figure 7.* Domain objects implementation.

The Request object serves as a container for the parameters being passed from the presentation component to the service component. This class encapsulates the Action performed by the user and the date and time when the request was submitted. Request is initiated by a presentation object and passed to an appropriate service handler via the Controller.

The Response object includes information about the business processing results (either local or remote) and the original Request object. The encapsulated Request object is used to fetch information about the original request in case of error. In addition, Response object has a responseCode attribute, which specifies if the processing was successful or not. In case of failure, the response object holds the information describing the error. Response object is created by a service handler and passed via Controller to the target presentation object.

The Errors domain object stores errors generated by PresentationObject's validate() method. These errors are generated within the presentation component and, as such, do not cross the presentation to service component boundary. Rather, user interface errors are intercepted by PresentationObject's commandAction() method and shown to the user.

All domain objects extend from an abstract DomainObject class, which hides the specifics of the underlying map data structure. This class provides convenient methods to store/retrieve properties, and to clear the internal data structure.

## 4.2. Presentation Component

mFrame application framework contains a single MIDlet object called MFrameApplication. The MFrameApplication class extends from J2ME's abstract class javax.microedition.midlet.MIDlet. startApp() method serves as an entry point of the application and is responsible to pass the thread of execution to the Controller which, in turn, invokes the first presentation object. PresentationObjects are always displayed by means of MFrameApplication's show() method.

PresentationObject abstract class serves as a template for the concrete view implementations. It is displayed on a per-screen basis and is primarily used to present a choice of actions and user interface components. PresentationObject implements J2ME's CommandListener interface. As shown in the sequence diagram (Figure 8), commandAction() method is invoked in response to a submitted event.

This method calls buildRequest() to instantiate the corresponding Request object. User interface validation is performed by means of validate() method enforced via IRequestValidator interface. If the validation fails, Errors object holding error information is passed to MFrameApplication, which displays a modeless dialog on top of the current presentation object. This dialog shows an error as name/value

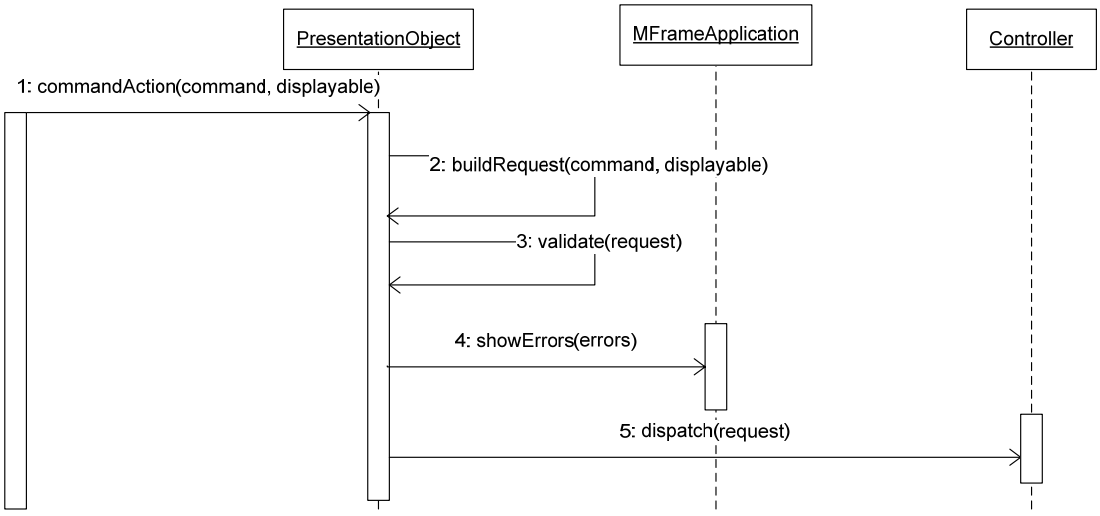



*Figure 8.* PresentationObject sequence diagram.

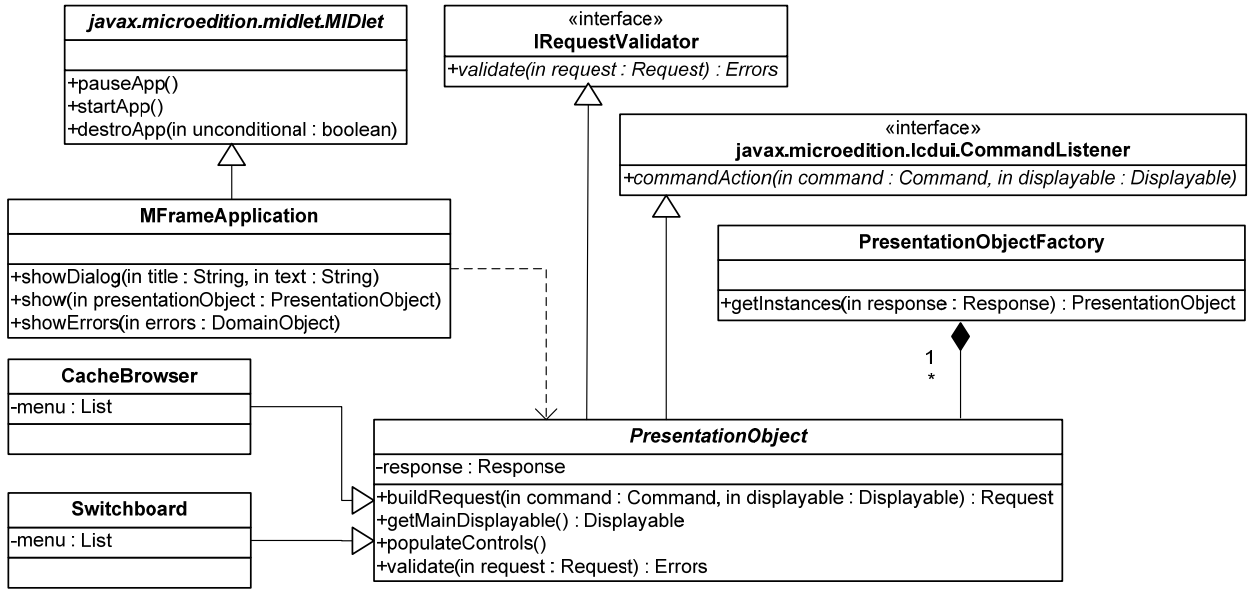



*Figure 9.* Presentation layer implementation.

pair. If the validation is successful, Request is dispatched via the Controller component.

The PresentationObjectFactory takes the Response received from the Controller, and creates an instance of the concrete PresentationObject. See Figure 9.

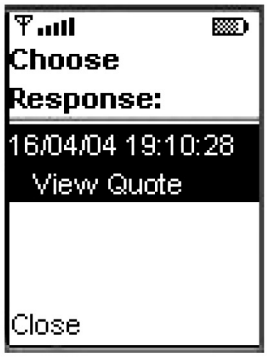
The framework provides two concrete implementations of PresentationObject: Switchboard and CacheBrowser (shown below). Switchboard is the first presentation object invoked upon the application initialization. It presents the list of available applications read from the configuration file as explained in Section 5.1.2.

CacheBrowser displays queued Response objects in the CacheHandler. This component displays information about the action performed by the user and the date and time when the original request was submitted. Upon selection of a Response from the list, the user is taken to the appropriate results screen.

## 4.3. Service Component

The ServiceHandlerFactory takes the Request received from the Controller and creates an instance of the ServiceHandler. ServiceHandlers implement process() method to perform business logic. In case of the local service implementation, this method invokes local data access objects, performs business logic and returns the results in the form of a Response object.


`Switchboard` Presentation Object




`CacheBrowser` Presentation

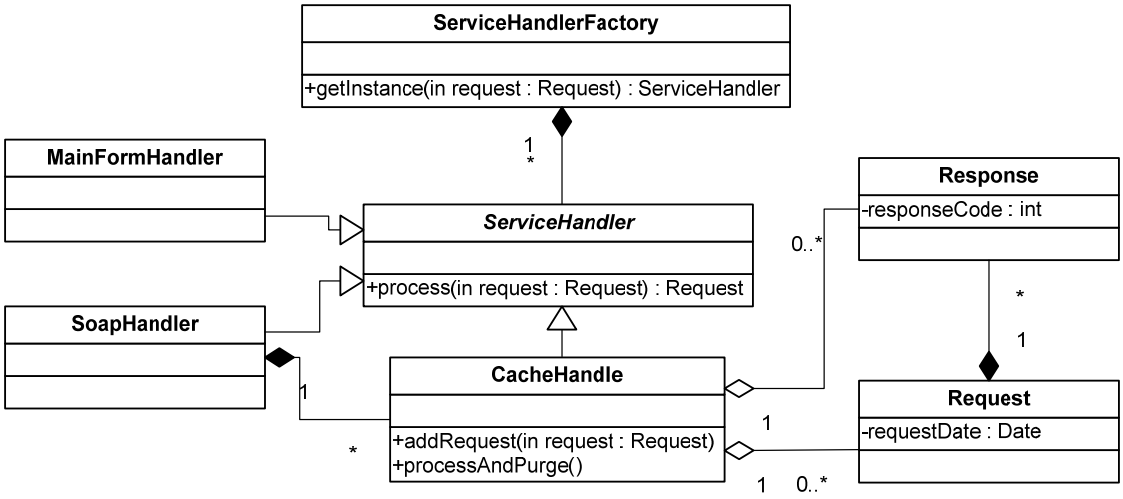



*Figure 10.* Model layer implementation.

SoapHandler encapsulates remoting details of the framework. It uses kSOAP libraries to build and parse XML messages exchanged between local application components and remote web service. SoapHandler converts input Request object into a corresponding SOAP request using kSOAP's getSoapRequest() method. Next, SoapHandler opens an HTTP connection to the external Web service and submits the SOAP message. Remote Web service parses the incoming SOAP request, converts it to server-side domain objects, and forwards it to middle-tier components responsible for business processing. Upon completion, Web service sends a response message over the same HTTP connection used to submit the request. SoapHandler reads the HTTP response stream and builds a SoapObject, which contains the SOAP response data. Next, the handler parses the response data using the SoapObject.getProperty() method. It is the responsibility of kSOAP libraries to automatically convert primitive SOAP elements into the appropriate Java types. Finally, these Java object are used to build the Response object that, in turn, is sent back to Controller or CacheHandler. See Figure 10.

The CacheHandler extends the ServiceHandler and encapsulates the collection of Request objects that are locally queued until the connection is re-established. The CacheHandler implements a worker thread to check the availability of the remote server, and to purge any expired Requests and Responses in the local cache. The cacheHandler uses an encapsulated SoapHandler to communicate with the external service. Whenever the external communication is restored, CacheHandler processes queued Request objects and queues Responses objects that are made available to the client application by means of the CacheBrowser presentation object.

MainFormHandler provides a list of available applications to be displayed by the Switchboard presentation object. This service object uses ConfigurationManager to retrieve the application list from the configuration file, as explained in Section 5.1.2.

## 4.4. Controller Component

This framework does not support multiple controllers – all the traffic will be handled through the single instance of the Controller object. Therefore, Controller is implemented as the Singleton pattern. The Singleton pattern is a reusable solution where the class itself is responsible for keeping track of its sole instance, while ensuring that no other instance of the object can be created, and providing a way to access the instance (Gamma et al. 1997).

As shown in the framework's sequence diagram (Figure 11), at the initial call, the Controller invokes the ConfigurationManager class that loads the information from the configuration file into an internal map structure. This information can then be retrieved by calling appropriate methods exposed by ConfigurationManager class.

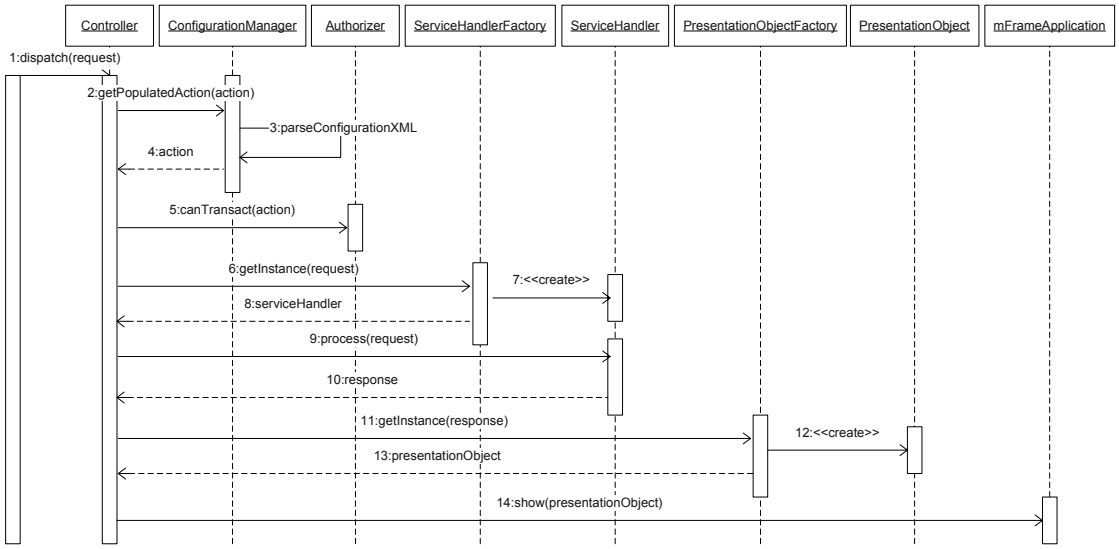



*Figure 11.* Controller sequence diagram.

Configuration file parsing is done using kXML pull-based APIs. In pull-based parsing (Sarang, Browne, Ayala & Chopra 2002), the parser intercepts an event and traverses the XML data while issuing callbacks to a previously registered event listener whenever it encounters particular structures in the data. This approach is very efficient since it does not have to build the complete document's tree in memory. ConfigurationManager is also implemented as a singleton class. See Figure 12.

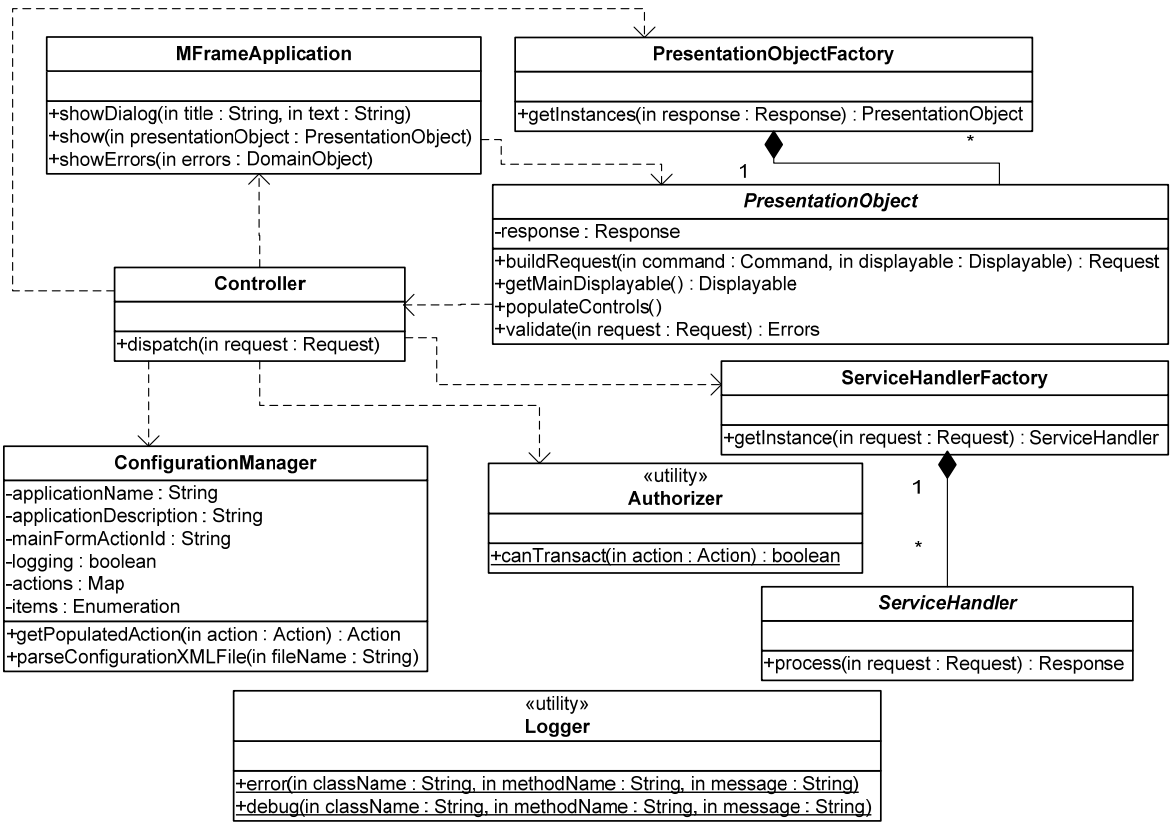



*Figure 12.* Overall implementation.

Controller class exposes a public method called dispatch(). This method sends the Request object to the corresponding ServiceHandler. Depending on the Response, an appropriate PresentationObject is instantiated, which is then passed to MFrameApplication's show() method to display processing results.

The Controller uses the Authorizer utility class to determine if the user is authorized to perform the requested business action. The Authorizer's canTransact() method is implemented on top of a device's role permission model. If such a permission model is not provided, the method returns true (i.e. the action is always authorized).

Finally, this framework provides a basic error logging service. Similarly to Controller and ConfigurationManager classes, Logger utility class is implemented as a singleton class. Logger's error() and debug() methods are implemented to write logging messages to device specific persistent storage. Logging information includes source class and method name information. In the mFrame, all critical exceptions thrown within the framework are caught and stack traces are logged to enable fault analysis.

## 5. A Case Study

In this section, a case study will be used to explain the Configuration and Client Implementation of an application using mFrame framework. The case study presents a prototype for a real application that was deployed later. An application will be constructed for a brokerage company that needs to access real-time stock prices and call clients with quote updates. Therefore, the application should fulfill the following requirements:

- Retrieve current quote for a specified stock symbol. The services.xmethods.net getQuote web service will be used for this purpose.
- Display a list of the stored phonebook entries. The phonebook entries will be retrieved from the local persistence storage.
- Resulting application has to be smaller than 128kB due to maximum mobile application download size.

### 5.1. Configuration

Configuration of mFrame application framework is done using a configuration.xml file. The configuration file has three major sections: application action descriptions, startup application list, and logging state. Following is the Configuration file specific to this application.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Application SYSTEM "configuration.dtd">
<Application name="Test" description="Testing Application">
  <Logging state="true"/>
  <MainForm actionId="switchboard">
    <Item label="Stock Quote" actionId="get_quote"/>
    <Item label="Phonebook" actionId="phonebook"/>
    <Item label="Cache Browser" actionId="cache_browser"/>
  </MainForm>
  <Actions>
    <Action actionId="switchboard" description="Switchboard">
        <CallbackSuccess presentationObject="org.mframe.view.Switchboard"/>
        <CallbackError presentationObject="org.mframe.view.Switchboard"/>
        <Process serviceHandler="org.mframe.model.MainFormHandler"/>
    </Action>
    <Action actionId="cache_browser" description="Cache Administration">
        <CallbackSuccess presentationObject="org.mframe.view.CacheBrowser"/>
        <CallbackError presentationObject="org.mframe.view.CacheBrowser"/>
        <Process serviceHandler="org.mframe.model.CacheHandler"/>
    </Action>
    <Action actionId="get_quote" description="Get Quote">
        <CallbackSuccess presentationObject="org.mframe.view.StockQuote"/>
        <CallbackError presentationObject="org.mframe.view.StockQuote"/>
```

```
    < /Action>
  <Action actionId="view_quote" description="View Quote">
            <CallbackSuccess presentationObject="org.mframe.view.StockQuoteResult"/>
            <CallbackError presentationObject="org.mframe.view.StockQuote"/>
        <Process serviceHandler="org.mframe.model.SoapHandler"/>
        <Parameters>
            <Parameter name="cache_request" value="true"/>
            <Parameter name="expire_in_milliseconds" value="300000"/>
            <Parameter name="url" value="http://services.xmethods.net/soap"/>
            <Parameter name="urn" value="urn:xmethods-delayed-quotes"/>
            <Parameter name="soap_action" value="getQuote"/>
        < /Parameters>
  < /Action>
    <Action actionId="phonebook" description="Phonebook">
            <CallbackSuccess presentationObject="org.mframe.view.Phonebook"/>
        <CallbackError presentationObject="org.mframe.view.Phonebook"/>
        <Process serviceHandler="org.mframe.model.PhonebookHandler"/>
    < /Action>
  < /Actions>
< /Application>
```

### 5.1.1. Action Description

As explained in the design and implementation sections of this paper, presentation and service objects are delegated at runtime. configuration.xml defines these objects in terms of <Action> XML element that describes: presentation object, service handler, and a list of optional application parameters. An <Action> element also includes a unique action id and description attributes:

- <CallbackSuccess> and <CallbackError> sub-elements identify presentation objects to be called in case of success and failure response codes. If needed, the same PresentationObject can be used for both success and failure callbacks.
- The service object responsible for handling the request is specified in the <Process> element.
- Parameter cache_request specifies that the request should be cached if connection is not available.
- Parameter expire_in_milliseconds defines duration for which the request will be valid
- Parameters url, urn, and soap_action describe the web service being invoked.

### 5.1.2. Startup Application List

As pointed out in Section 4.2, Switchboard object is the first user interface component displayed upon the application initialization. It presents a list of available top-level applications that the user can choose from. This list is defined by means of the <MainForm> element. <Item> sub-element defines a top-level application, including the label which is presented to the user, and the corresponding actionId. Application items are displayed in the same order as they appear in the <MainForm> element. If Switchboard does not fulfill requirements of the mail form component, a different action value can be used.

### 5.1.3. Logging State

For the performance and storage limitation reasons, mFrame introduces a check compared before the actual logging, which defines if the logging should be executed. Therefore, an overhead of the data manipulation and persistence I/O activity can be controlled. This check is configured within configuration.xml file, as follows: <Logging state="true"/>

## 5.2. Client Implementation

As described in the previous sections, an Authorizer can be implemented optionally. This is typically a job of framework developers and constitutes the first step in the conception of the application. The second step is the implementation of presentation and service objects and is the responsibility of application developers.

### 5.2.1. Authorizer

The Authorizer's canTransact() method can be optionally overridden to customize to a specific cellular device. The default Authorizer implementation returns true value.

### 5.2.2. Presentation Object

The steps towards implementing a presentation object will be demonstrated by the means of StockQuote object, shown in Figure 13. First, user interface components are created and initialized:

```
private Form form=new Form ("Stock Quote");
private TextField symbolField=new TextField ("Symbol", "", 5, TextField.ANY);
Command getCommand=new Command("Get", Command.SCREEN, 1);
```


*Figure 13.* StockQuote presentation object.

Next, the user interface controls are populated and positioned using populateControls()

```
public void populateControls() {
    form.append(symbolField);
    form.addCommand(getCommand);
 form.setCommandListener (this);
}
```

This method registers with the command listener responsible for events fired by the getCommand component. Normally, the populateControls() method would retrieve information from the Response object and populate controls, but in this example we are building a presentation object without any input values.

The buildRequest() method sets the internal Request object that is passed to Controller for further processing. Request object is built with view_quote action id and the parameter named Symbol:

```
protected Request buildRequest(Command command, Displayable displayable) {
    Request request=new Request();
    Action action=new Action();
    action.setActionId("view_quote");
    request.add("Symbol", symbolField.getString());
    request.setAction(action);
    return request;
}
```

Finally, validate() method is implemented to include any presentation specific validation logic. In this example, information has to be entered in the symbolField field before it is passed to SOAPHandler for remote processing:

```
public Errors validate(Request request) {
    Errors errors = new Errors();
    if (symbolField.getString()==null ||
        symbolField.getString().length()==0) {
        errors.add("Symbol","Cannot be empty");
}
        return errors;
}
```

This method takes the populated Request object and performs validation on it. Any errors identified in the validation are registered in Errors collection and returned to the client. StockQuote uses SoapHandler to retrieve quote value, and as such does not require custom handler implementation. Therefore, the last step to complete this business service would be to build PresentationObject to display the stock's value.

### 5.2.3. Service Handler

Business logic is defined by means of the service handlers. Method process() performs business logic and sends the results in the form of a Response object back to the presentation layer via the Controller object. Here is an example of PhonebookHandler's process() method:

```
public Response process(Request request) {
    Response response=new Response();
    PhonebookDataAccess phonebook=new PhonebookDataAccess();
    response.add("phonebook", phonebook.getPhonebook());
    return response;
}
```

This method uses PhonebookDataAccess mock data layer object to create static enumeration of phonebook entries. PresentationObject's populateControls() method presents the resulting list of phonebook entries:

```
public void populateControls() {
    Vector phonebook=(Vector) response.get("phone-
    book");
    for (int i=0; i < phonebook.size(); i++) {
        String phone=(String) phonebook.elementAt(i);
        menu.append(phone, null);
    }
}
```

Significant cost and effort savings were achieved due to:

- Simplified navigation management via configuration file – the configuration file allows centralized setting and makes it easy to switch to a different operation mode.
- Convenient application packaging and deployment – the simplicity (both in terms of interface as well as implementation) allows easy deployment.

The resulting application size is 69.7kB and satisfies the size requirement.

## 6. Conclusion and Future Work

mFrame is an innovative, reliable and extensible framework that supports rapid application development for compact mobile devices. mFrame enables dynamic presentation and service management where objects are registered by means of an external configuration facility provided by the architecture. Thus, new business services can be easily added. This framework introduces message queuing between compact mobile devices and external service providers. This caching model enables uninterrupted application execution, even though a communication channel could be lost due to resource preservation or remote service unavailability. However, this framework primarily focuses on flexibility and reusability, and hence applications built on top of it may perform slower than conventional applications.

Future work would include improving the validation mechanism such that validation routines are applied to the corresponding fields via the configuration file instead of specifying them in the validate() method. If form values are missing or in an incorrect format, the current form would be automatically redisplayed with error messages, which is defined in the configuration file. Another improvement is to implement the data access layer to enable interaction with a mobile device's persistence storage. The data access package would include classes to execute query procedures and retrieve result sets from the storage. This research did not prescribe any rules about the external Web service implementation. It is up to the server developers to implement these services and corresponding business objects and deploy them to the application server. One of the architecture's future improvements will be development of the server Web service-based framework that will transparently map mobile application's request objects to underlying remote business services.

## References


[1] ALUR, D., CRUPI, J., MALKS, D. *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall, NJ, June 2001.

[2] BALAKRISHNAN, H., SESHAN, S., KATZ, R., Improving reliable transport and handoff performance in cellular wireless networks, *Wireless Networks*, Vol. 1, No. 4, December 1995, pp. 469–481.

[3] BELLAVISTA, P., CORRADI, A., STEFANELLI, C., Mobile Agent Middleware for Mobile Computing, *IEEE Computer*, Vol. 34, No. 3, March 2001, pp. 73–81.

[4] BOURGES-WALDEGG, D., DUPONCHEL, Y., GRAF, M., MOSER, M., The Fluid Computing Middleware: Bringing Application Fluidity to the Mobile Internet. *Proceedings of the 2005 Symposium on Applications and the Internet*, Trento, Italy, 2005.

[5] CARNELL, J., LINWOOD, J., *Professional Struts Applications: Building Web Sites with Struts, Object Relational Bridge, Lucene, and Velocity*, Wrox Press Inc., Birmingham, UK, March 2003.

[6] Cocoon Application Framework, `http://cocoon.apache.org`

[7] EDWARDS, W., MYNATT, E., PETERSEN, K., SPREITZER, M., TERRY, D., AND THEIMER, M., Designing and implementing asynchronous collaborative applications with Bayou, *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pp. 119–128, Banff, Alberta, Canada, 1997.

[8] GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., *Design Patterns: Elements of Reusable Object-oriented Software*, Addison-Wesley Professional Computing Series, MA, 1997.

[9] Grails Application Framework. `http://grails.codehaus.org`

[10] jShrunk Application Framework. `http://jshrunk.sourceforge.net`

[11] LARMAN, C., *UML and Patterns: An Introduction to Object-oriented Analysis and Design, and the Unified Process*, 3rd edn, Prentice Hall, NJ, July 2001.

[12] LUDWIG, R., KONRAD, A., JOSEPH, A., Optimizing the end-to-end performance of reliable flows over wireless links. *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 113–119, Seattle, Washington, 1999.

[13] Maverick Application Framework. `http://mav.sourceforge.net`

[14] MÜRK, O., KABANOV, J., Aranea: web framework construction and integration kit. *Proceedings of the 4th International Symposium on Principles and Practice of Programming in Java*, Vol. 178, pp. 163–172, Mannheim, Germany, 2006.

[15] SARANG, P. G., BROWNE, C., AYALA, D., CHOPRA, V., *Professional Open Source Web Services*, Wrox Press Inc., Birmingham, UK, July 2002.

[16] SATYANARAYANAN, M., Fundamental Challenges in Mobile Computing. *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 1–7, ACM Press, 1996.

[17] Spring Application Framework. `http://www.springframework.org`

[18] VARSHNEY, U., Multicast Support in Mobile Commerce Applications, *IEEE Computer*, Vol. 35, No. 2, February 2002, pp. 115–117.

[19] WELLING, G., BADRINATH, B., An Architecture for Exporting Environment Awareness to Mobile Computing Applications, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 391–400, May 1996.

*Received:* October, 2006
*Revised:* September, 2008
*Accepted:* January, 2009

*Contact address:*
Hossein Saiedian, PhD
Professor and Assoc Chair
Department of EECS
School of Engineering
University of Kansas
Lawrence, KS 66045, USA
Tel: +1 785-864-8812
e-mail: `saiedian@eecs.ku.edu`
URL: http://people.eecs.ku.edu/ saiedian

HINKO VINCAR is an Information Systems Officer at the United Nations (in Cologne, Germany). He received a Master's degree in computer science from the University of Kansas (USA) in 2004 and a Bachelor's degree in computer information systems from the Ottawa University (USA). Before joining the United Nations, he was a software engineer and systems architect at the YRC Logistics.

HOSSEIN SAIEDIAN is currently a professor of software engineering in the Department of Electrical Engineering and Computer Science at the University of Kansas (KU) and a member of the KU Information and Telecommunication Technology Center (ITTC). Professor Saiedian's primary area of research is software engineering. He received a Ph.D. in computer science from the Kansas State University (USA) in 1989.

SERHIY MOROZOV is currently working on his Ph.D. in computer science at the University of Kansas (USA). He received a Master's degree in computer science from the University of Kansas (USA) in 2007 and a Bachelor's degree, also in computer science, from the Westminster College (USA) in 2003.