

Mobile Services

2008/2009

Project's report

Read-J2-Me

June 9, 2009

Anton Dignös
Omar Moling

Project's goal

The application should allow users to subscribe to news feeds using the mobile phone and Java Mobile. The application should allow the user to subscribe to new feeds, remove a subscribed feed and browse a subscribed feed. When browsing a feed, the user should be able to open a news item's content, update the news items of the feed, search the news items by a string in the title. The user should have the possibility to tag news items with one or more tags that were defined by the user. When any tag has been defined, the user should be able to filter the news items according to the selected tags.

In addition, application settings may be saved and automatically loaded at the next start-up of the application, as i.e. the maximum number of news items that should be stored by each saved feed. Tagged news items should be prevented from automatic deletion when an update is being performed. Adding a tag to a news item should make it to be ignored from the automatic deletion that runs when the feed is updated and the maximum number of items is reached.

Application's functionalities

The Read-J2-Me application allows users to subscribe to news feeds that are published in the Atom syndication format. For what concerns the subscription to feeds, the application provides the following operations:

- subscribe to a new feed
- delete a feed's subscription
- update a feed's news items
- delete a news item
- mark news items as unread
- tag news items

An Atom news item defines several information. The Read-J2-Me application reads, at this stage, a subset of these attributes:

- id (for unique identification)
- title
- published
- updated
- link
- content
- summary

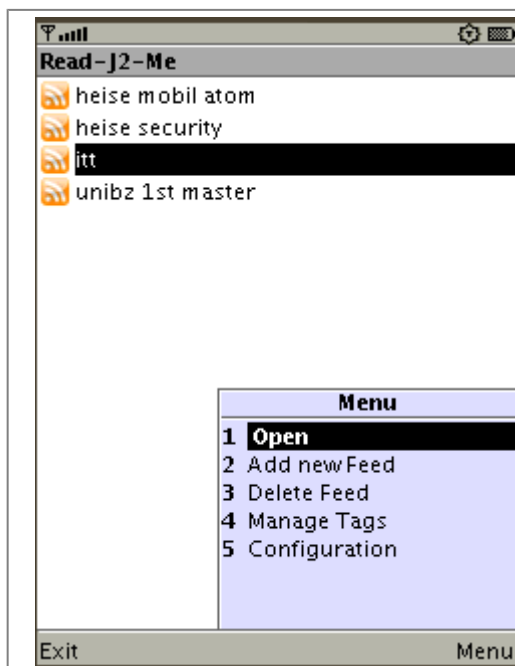
Locally, for each news item, the assigned tags and the item's read-status will be saved so to allow a different visualization of read and unread, and tagged items. The feed to whom the news items belong to will also store all item's IDs that were read so to avoid storing again news items that were deleted by the user.

Tags can be defined by the user itself and the name has to be unique, otherwise the user is informed to choose another name. Tags can be added either from the tag's management view, where they can also be deleted, or when tagging a news item, in the case the desired tag is not already available.

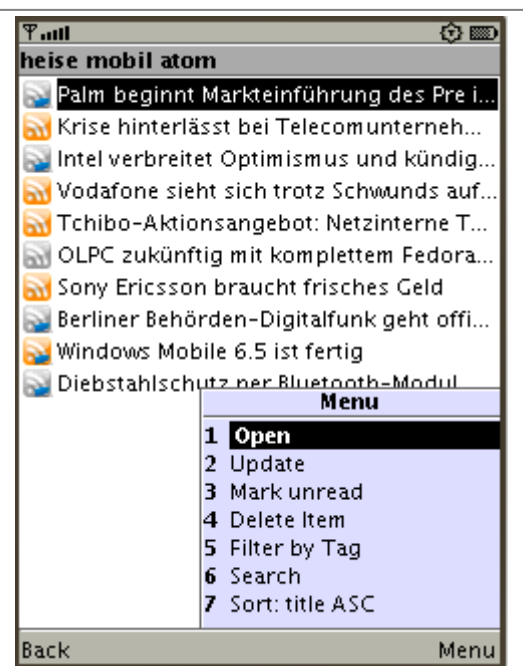
The application's configuration can be changed by the user and will be saved on the local record store. This configuration holds, at this stage, the settings about the maximum number of news items a feed is allowed to store.

User interface

Special focus was set on the usability of the application. The commands present in the views are ordered according to what may be the common usage order for a common user, so to have the most used command the first to be selected when in the case there are more than 2 commands in the view and the "menu" opens. The application's general functionalities, namely the configuration and the tags management are made available to the user in the first view that appears, which also displays all feeds the user subscribed yet.



Screenshot 1







Screenshot 2

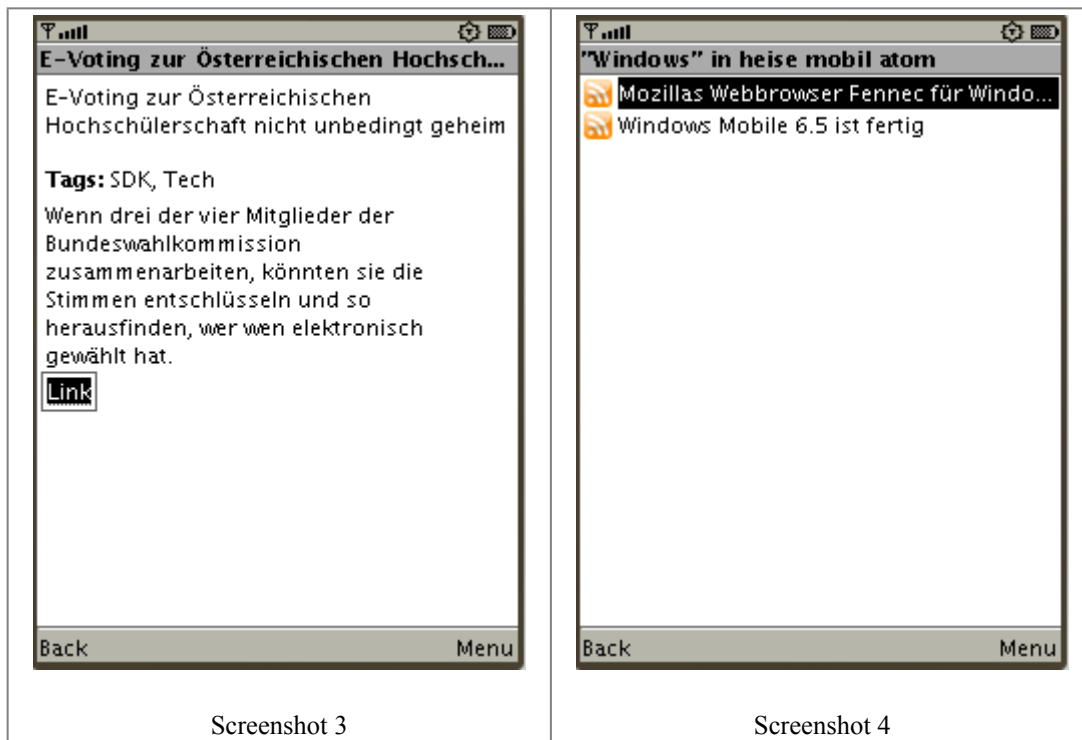
Screenshot 1 shows the first view that the user will see when the application starts. This view contains all feed's subscriptions and allows the user to open a feed, add a new subscription to a feed, delete a subscription to a feed. Moreover, since this is the first view, the user finds here also the commands to enter the tag-management and the configuration settings.

Screenshot 2 shows the list of news items of a feed. The list shows all news items since the application's configuration defines a limit for the maximum number of news items stored by each feed. The news items have an icon on the left-hand side that is thought to inform the user about whether the specific news item has already been read or not, and whether it has been tagged or not. In addition, the user can select to update the feed, mark a news item as unread or delete a news item. The user has also the possibility to filter the news items by one or more tags and to perform a search on the news items by the title. For easing the search of specific news items, like i.e. in a calendar feed where date and time are set at the beginning of the title, the user can select to order the news items by the title, in either ascending or descending order, or to go back to the default order by date, which displays the most recent news items first. Actually the command to order by title in descending order is available only when first the order by title in ascending order is selected, and in one of these two cases, the command to order by date is made available again. This to prevent having too many commands in the menu.

The icons used and their meaning are illustrated in the following table:

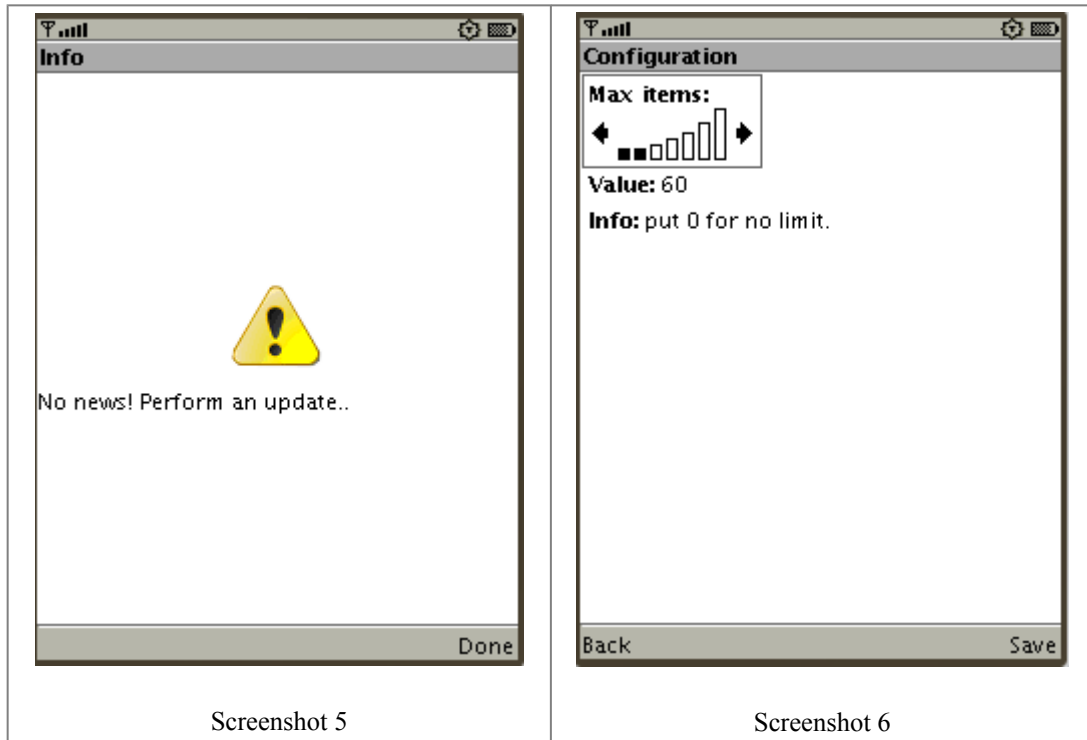
	read	tagged
		
	✓	
		✓
	✓	✓

(note the blue tag on the latter two icons in the bottom-right corner)



Screenshot 3 shows the detailed view of a news item's content. This view contains all elements that were defined and recognized in the XML data. Starting from the title, the view then shows the tags the news item was tagged by, if any, the summary or content, if any, and a link to the news item's link, if any. To this link, a command is associated which calls the MIDlet method *platformRequest* with the provided URL. This leads the mobile phone to either start the the local browser immediately and pause the Read-J2-Me application, or, in the case the browser can be started only at the end of the execution of this application or there is no browsing capability provided by the mobile device, the user is informed with a warning (which will be shown in *Screenshot 5*) about this. Moreover, from the menu, the user can select the command to add tags to the opened news item.

Screenshot 4 shows the searching results for the string "Windows" in the feed "heise mobil atom". The search string is displayed in the title bar of the List and all matching news items are listed in the list. The results-view for the filtering by tags is equal except for the fact that instead of a search string, the tags selected for filtering are displayed in the title bar of the List.



Screenshot 5 shows a warning alert used to inform the user about i.e. in this case the absence of news items to be opened and that an update of the feed should be performed. This kind of warning is also used everywhere where some input by the user is required and is missing. There exists another similar alert, a so-called error alert which

Screenshot 6 shows the Form for editing the application's configuration. Here the user can set the maximum number of news item a feed should be allowed to store. When this limit is exceeded, the less recent news items are deleted until the limit is reached again. Tagged news items are prevented from this automatic deletion. This also means that if there are more tagged items that the maximum number of allowed news items, that may happen in case the limit is set in second point in time to a lower value, that feed will hold more news items, namely all tagged items.

Tasks that may take a longer time to execute add a Ticker to the view to inform the user about the ongoing operation. This is especially the case for the feed's update operation whose duration depends also on the data connection speed, and for intensive memory operations, like i.e. the deletion of a tag, which causes a scan of all news items of all feeds to remove the tag being deleted.

Implementation details

During the start-up of the application, the local record store is checked if a configuration of the application settings has been saved before. If this is the case, the saved configuration is loaded and made available to the applications functionalities, otherwise the default settings will be used.

Exceptions that may be caused by the input/output operations, parsing of XML data, reading and writing from the local record store and others are caught everywhere they can occur and the user is provided with some information about what happened. The application then continues at the last state not affected by the exception.

Objects that can be stored on the local Record Store extend the IPersistable interface that defined the two methods needed to code and decode the object in a byte sequence. Some of these objects (i.e. Feed, NewsItem) store also the Record-ID where they are stored. This ID can be retrieved just before the objects are going to be stored. This allows the application to avoid i.e. the usage of a filter, that would go through all records in the RecordStore, to find the corresponding record of the feed/news item in case of deletion.

It follows a detailed description of all classes, and the relative responsibility, in the application, ordered by package (it.unibz..) and by name:

Class Name	Responsibility
<i>.readj2me</i>	
ReadJ2ME	Main class, extends MIDlet, start the application by settings up the first view and loading the stored configuration, if any. Makes needed system methods available to other classes (i.e. MIDlet.platformRequest(String URL)).
<i>.readj2me.controller</i>	
ImageLoader	Loads images performing a check on the path
Networking	Opens a connection and makes the content available as InputStream (secure connections are supported)
NewsItemSearchFilter	Filter used to filter out news items having the specified string in the title
NewsItemTagFilter	Filter used to filter out news items tagged with the specified tag
PersistentManager	Contains all methods that directly access the Record Store
XmlReader	Parses an Atom-XML InputStream to objects of type NewsItem
<i>.readj2me.controller.comparators</i>	
FeedComparator	Comparator used to order Feeds alphabetically
NewsItemDateComparator	Comparator used to order news items according to the Updated-date, in either direction
NewsItemTitleComparator	Comparator used to order news items alphabetically according to the title, in either direction
TagComparator	Comparator used to order Tags alphabetically
<i>.readj2me.model</i>	
Configuration	Application's configuration
Constants	Static Final 'variables' for commonly used strings and of known images
Feed	A feed
IPersistable	Interface defining the methods an object has to implement to be coded/decoded for the Record Store
NewsItem	A news item
Tag	A tag

<i>.readj2me.view</i>	
ConfigurationForm	Form to modify the application's configuration
ErrorAlert	Alert used to inform the user about a system error
FeedForm	Form used to add a subscription to a feed
FeedList	List showing all subscriptions to feeds
InputForm	Abstract class that holds logic used for views that require input from user
NewsItemForm	Form that displays the content of a news item
NewsItemList	List showing all news items of a feed
NewsItemSearchForm	Form used to let the user define the search string on news items
TagChoiceFilterForm	Form used to let the user choose the tags to filter the news items by
TagChoiceForm	Form used to let the user choose the tags to tag a news item
TagForm	Form used to add a new tag
TagList	List showing all tags for tag-management
WarningAlert	Alert used to inform the user about warning like i.e. missing inputs, no feed to open ..

Some classes are defined as singleton classes so that only one instance can be created. It is the case for the PersistentManager, XmlReader and Configuration classes. Some code contained in the Networking class was taken from the course's materials and adapted to the application's needs.

UML Class Diagrams

The following class diagrams show, for the most important packages, the classes and the methods of the application's implementation.

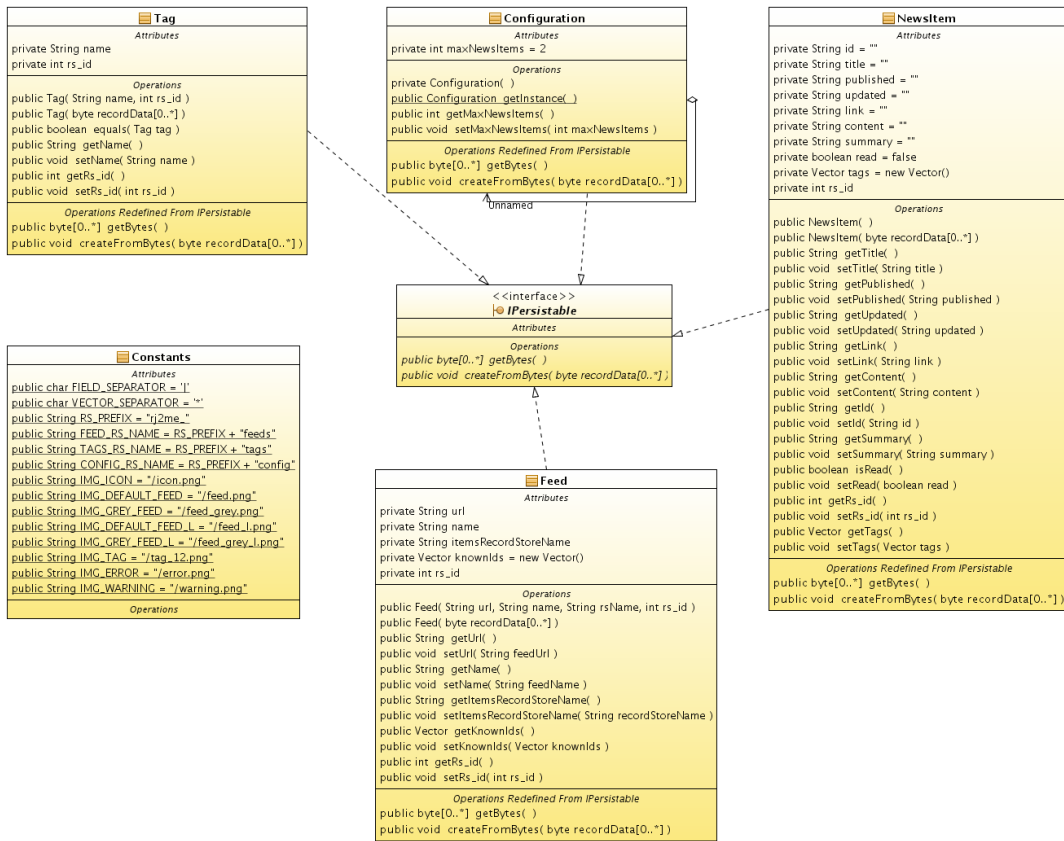


Diagram 1

Diagram 1 shows the UML class diagram for the *it.unibz.readj2me.model* package.

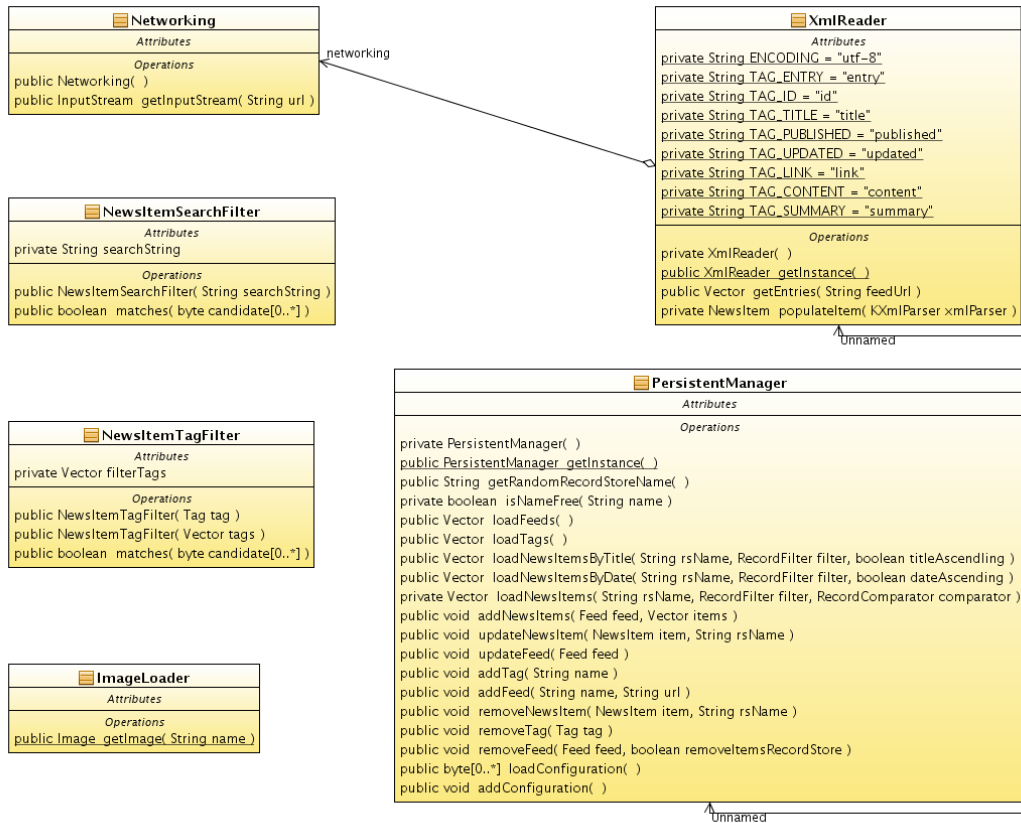


Diagram 2

Diagram 2 shows the UML class diagram for the *it.unibz.readj2me.controller* package.

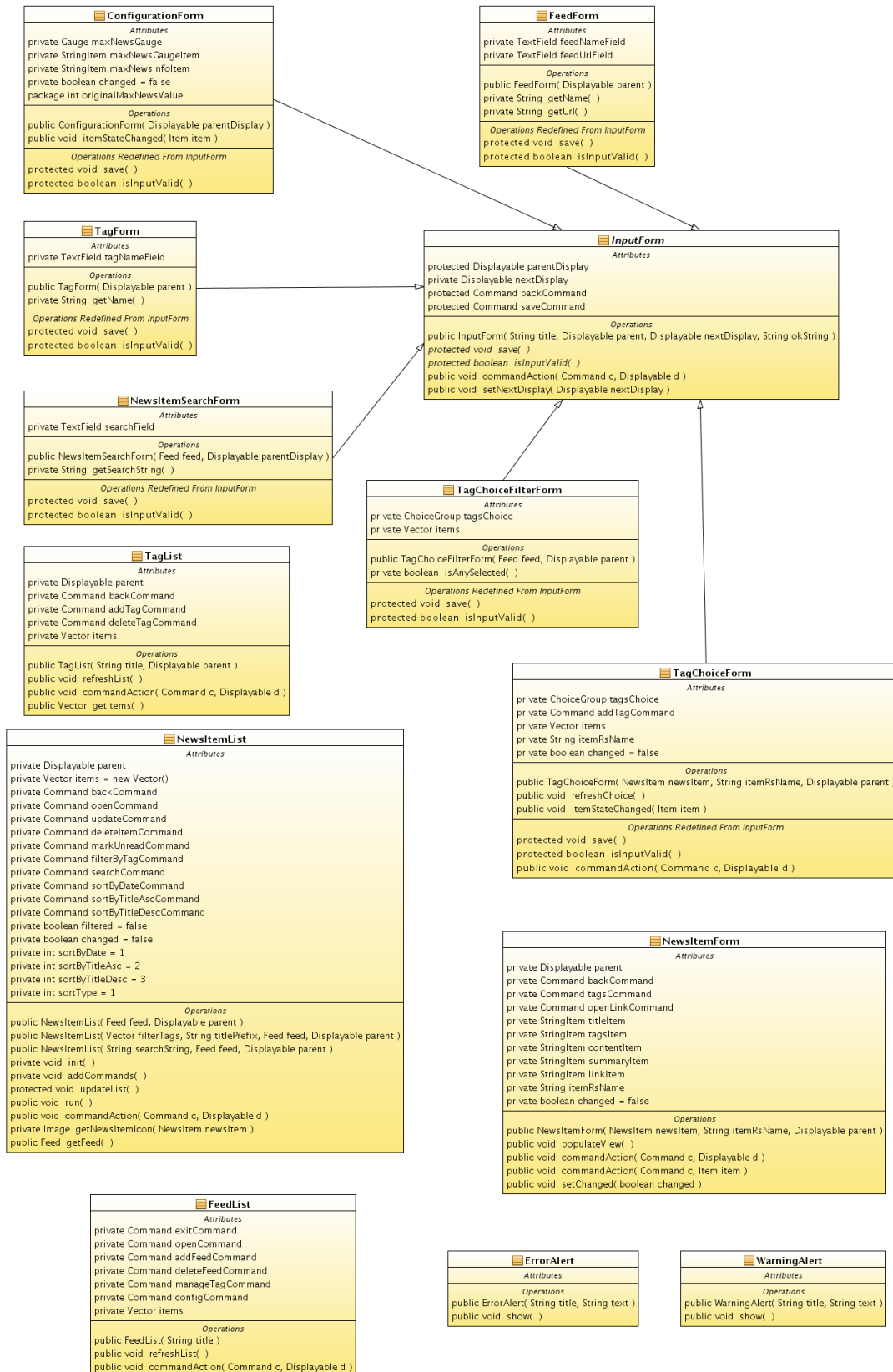
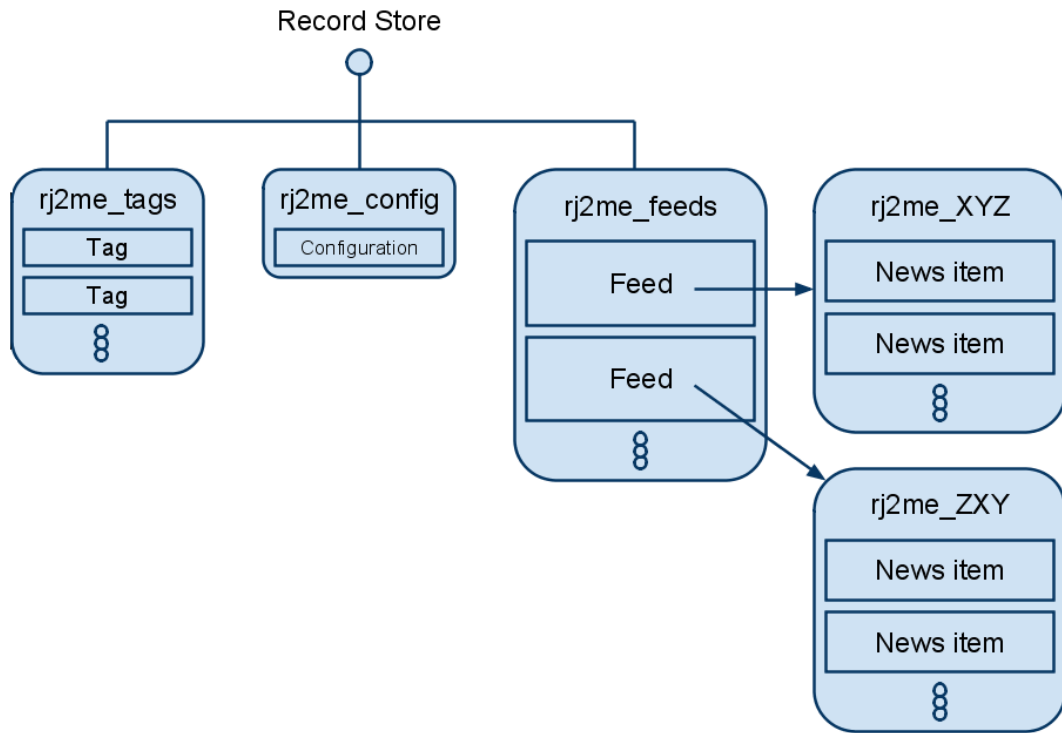


Diagram 3

Diagram 3 shows the UML class diagram for the *it.unibz.readj2me.view* package.

Management of the Record Store



Schema 1

As *Schema 1* represents, the management of the record store is build as follows: there are three record stores known in advance by the Read-J2-Me application. The left-most record store stores, in each record, a tag object, containing all necessary data and also the record-ID for improving the performance while i.e. a deleting operation. The next record store will contain only one record, which represents the application's configuration. Only one record is needed since there shall by only one configuration saved. The next record store stores the feeds, with all necessary data, the record-ID for better performance and in addition also the name of the record store containing the news items that belong to the feed. This record stores are represented near the right-most border of the schema, and store all news Items of the feed with all necessary data and the record-ID. The application will own at most $3 + x$ record stores, where X is the number of feeds or records in the "rj2me_feeds" record store. It has been decided in design phase to adopt this schema for managing the record stores, since operations like ordering and filtering of the records inside a record store perform the necessary checks on all records. By splitting the news items of each feed in separate record stores, the performance of such operations is improved and also in the case of browsing one feed's news items, the need of filter out the feed's news items has disappeared.

Technical problems and solutions

Most technical difficulties were encountered in the parsing of XML data. A light-weight pull-parser, namely kXml, has been used. When testing the application with several Atom feeds, some feeds could not be read and parsing exceptions were thrown until the cause could be detected and a solution implemented. Some feeds start the XML with a space character, which the used pull-parser could not read. By removing the first characters until the effective starting of the XML data, the parser could work without any interruption. It took some time to find a solution since the cause of the exception was not clear at beginning.

Another problem we encountered concerned the performance of the local storage mechanism. Firstly we decided to give to each feed a separate record store for the news items, so to limit the operations performed by i.e. record filters to a subset of the news items stored by the application. This requires to store inside the feed the name of the record store containing it's news items. This names are generated randomly starting with a predefined prefix. The additional space needed by the feed object to be stored locally may be negligible considering the performance gain induced by this architecture. Secondly, since all news items (or as many as defined by the user in the application's configuration) are stored locally by the application, specific operations on these records got a bad performance when the number of records stored increased. I.e. the deletion of a single news item required the usage of a filter to find the intended record. In the case the corresponding feed owned one hundred records, the filter would check all one hundred news items by first building the object from the byte data of the record and then performing the check whether it was the intended news item. The solution we have applied consists in storing inside the object itself the ID of the record where it is stored. This ID can be retrieved right before the record is being stored. When the object should be then deleted from the record store, the application simply calls the method for deletion of a specific record, identified by it's ID. This allows a greater performance when deleting or modifying stored objects.

Additionally, exception handling had to be worked out to find a common way to handle all exceptions. All exceptions are caught where they cause the operation's flow to be unable to continue, so that no operation that is affected by the exception continues. On the other side, all operations done till the moment the exception was thrown, and that should be reversed, are reversed in the *catch* code section in order to avoid inconsistent behaviors. The user is informed in such cases by an alert which may be a warning or an error alert, depending on the exception's seriousness. An example may be the catching of record store exceptions, like i.e. full-memory, when storing the application's configuration, which first updates the configuration object for the running application and then updates it on the record store. If something goes wrong while performing this update, also the running application's configuration is reverted to the previous status in order to avoid having the new configuration while running and a different (the previous) by the next start of the application.

A bug in the MIDP emulator of the WTK (version 2) caused some problems¹. To mark unread news items differently, in addition to the different icons used, we wanted to set the list entry's font as bold. This property is dependent on the device's implementation, but in case it is supported, the list entry would have been formatted with a bold font. After having used the *setFont* method on the list at least once, the WTK emulator does not allow further additions to the list. Whenever a feed was updated after at least one entry of the list had been modified by means of i.e. settings the news item unread or read one news item, the application threw an exception if new news items had been found and were tried to be added to the list. Since the application was tested on both real devices (on Windows Mobile and Nokia devices) and the emulator primarily, but primarily on the emulator, we decided to

1. <http://discussion.forum.nokia.com/forum/showthread.php?t=160403&page=2>

drop the functionality of marking unread news items in bold to avoid this exception to be thrown because of the emulator's bug.

To avoid the repetition of code portions among several views that are required some input by the user, an abstract class was defined. This class, named *InputForm* (can be seen in *Diagram 3*), provides basic functionality required by such a view, namely a save/confirm method which is called if the input is to be considered valid. The actual implementations of the two methods have to be defined by the classes that extend this class. This way, the commands used and the command listener are defined only once except for particular behaviors.

Future work

We could reach the goal of this project by developing this application which allows the subscription to Atom feeds and it's management, the storage of all news items and the management of these, like i.e. the tagging functionality, single deletion, read-status change etc.

As future work, the application may be extended to read more/all attributes of an Atom feed, like i.e. downloading the (optional) image a news item may specify, store it locally on the mobile phone and displaying it when the user decides to display the news item's content. Moreover, parsing capabilities for the RSS syndication format may be added so to become a more flexible feed reader.