Supporting User Generated Content for Mobile News Services: A Case Study

Regular Paper

Christos K. Georgiadis

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
* Corresponding author E-mail: geor@uom.gr

Abstract Web 2.0 applications encourage users to contribute to the production of richer content. In this context, our work mainly focuses on providing mobile users the ability to share content and to support user generated content production. Specifically, in our case study, features both for a mobile Web and for a mobile native application are implemented, capable of providing news services enriched with indicative social networking elements. The results of our work are largely related to the understanding of the required proper solutions, based on the investigation of serious technical challenges: the XML-RPC library for the Android platform is exploited, as well as a specific Backend Joomla! component is built (Rsstoa) to handle consistently external content sources, such as feeds and multipart emails.

Keywords Mobile News Services, News Infomediaries, User Generated Content, Mobile Web

1. Introduction

The Web has fostered the emergence of 'news infomediaries', which are applications/sites that work in publishing material already published by a third party, acting as specialists in information brokering [1].

Characteristic examples are the portals of popular Internet service providers (such as MSN–the Microsoft Network) and the news aggregator services (such as the Google News service). In the latter, advanced features for searching articles and sorting/grouping the results are present along with the provision of hyperlinks to news messages from different content websites together with the first two or three lines of the concerned message [2].

Moreover, Web and mobile users make increasing demands on the role of users-authors: they seek flexible ways to add content of various forms as well as to categorize and rate content. Consequently, the objective of this work was to facilitate the involvement of mobile users to produce and upload new content on the news website and to facilitate their participation in the promotion and evaluation of the existing content. An important aspect of the provided functionality is the capability to pull news content from different sites and present it in a mobile environment. A significant addition was the design and implementation of a specific component for the automatic storing of articles derived from external sources, such as RSS/Atom feeds and emails.

Website creation was based on the popular Joomla! Content Management System (plus the mobile Joomla!

component). Some modifications, adaptations and additions were applied in order to enrich it with functions necessary for the needs of the participative Web. Two discrete ways for accessing the content through mobile devices are described. The first concerns the adaptation of content (depending on the device specifications) that is performed by the website, whereas the second concerns the development of a suitable customized mobile application that allows users to access the content of the news website. For this reason, an application for the Android platform was developed in the NetBeans environment. Our approach adopts the two following critical success factors for content sharing and social network services (in the context of convergent mobile and Internet technologies, as they are identified in reference [3]:

- Focusing on open source code: relative observed benefits include simplified integration with other services, savings on proprietary technology and software acquisition, improved visibility and low barriers for external developers.
- Focusing on the integration with other social applications: relative observed benefits include increased service usage and expanded channels for new user acquisition.
- Leveraging on previous technological know-how and intangible assets: relative observed benefits include establishment of key business partnerships, new user acquisition, positive reputation and technical stability and scalability.

After using the news website under real conditions, an evaluation of its usability was performed, both by selected users and by random users who were forwarded to the Website through popular social networks or who discovered the Website themselves through search engine results. This evaluation was of vital importance, as the feedback received allowed correction of bugs and implementation of additional features. The results of our work are primarily related to the understanding of technical challenges and the development of proper solutions regarding the support of the "participative relation" of mobile users with the news website. In addition, an important aspect of this work is the resulting conclusions regarding the provided mobile news services that could be used as an important decision making tool by news and media professionals. Finally, we have to mention that some basic ideas relative to initial designs were presented in reference [4].

2. News Services

Analysing typical user requirements on news-oriented websites, we may distinguish the following user categories: visitor, member, author and administrator.

Visitors (the unregistered users) have read access rights mainly. Because our focus is to support social networking, we may allow them to share news content (articles) that they like via popular social networking sites (such as Facebook) and to rate news content.

Visitors may become members either by registering using our registration service or by certifying their identity via a social network. Members have extra access rights: they may post comments and receive newsletters by e-mail.

On the other hand, authors have content management rights: they may upload/withdraw/publish/delete news, organize their articles in categories, parameterize the presentation of their articles and manage the comments on their articles. They may publish articles through RSS/Atom feeds and via emails. Finally, they may upload multimedia for their articles from a variety of resources including popular social networks (such as YouTube and Picassa).

Syndication is currently a major and important part of the Internet. There are two main forms of Syndication: RSS 2.0 (Really Simple Syndication) and Atom 1.0 (Atom Syndication Format). Although they accomplish the same objectives there are some differences. RSS 2.0 is a standard copyrighted by Harvard, while Atom 1.0 has been defined in RFC4287. Atom was invented because the RSS 2.0 prototype was considered a prototype with limited expansive opportunities and capabilities [5]. Websites that provide RSS and ATOM feeds can maintain an «open connection» with their users. That means users do not have to periodically visit a website as they can simply use the aforementioned feeds to stay informed. Therefore, our approach for providing news services supports authors importing articles using RSS/Atom feeds. In this way, all users (members and visitors) may have access to content provided by syndication feeds.

3. Mobile News Services

Increased sophistication of mobile technology makes itself an ideal channel for offering valuable services to mobile users [6]. The following factors should be taken into consideration:

• A huge variety of mobile devices exists and these devices differ (from a developer's perspective) in at least two major parameters: the screen size and the reading format.

• Internet access is required generally and not necessarily in a constant online state. However, certain alternatives of accessing Internet content may be used: (a) the Mobile Portal model refers to browser-based approaches that allow users to access existing website content (the mobile Web) and (b) the Mobile Application Store model refers to native application implementations.

3.1 Web Technology

Most mobile devices have a preinstalled Web browser that allows users to access websites. But restrictions are applied because of the screen size and the limited capability of typing text through a touch-screen, which imposes selective modification of appearance of content depending on the specifications of the mobile appliance that is being served by the website. Such websites are either developed separately from their associated desktop-oriented version or both the versions are created simultaneously by employing methodologies that support multi-platform context-aware websites requiring an extensive engineering effort [7]. Context-aware adaptation for web browsing on handheld devices would be one of the major new functions of smart handheld devices in the near future [8].

3.2 Mobile Native Application

An additional application is developed tailored to a specific mobile platform, aiming to cover complete user requirements by exploiting the advanced characteristics of the current mobile device. This approach, despite the apparent weakness (building a different application for each platform is very expensive if written in each native language), has a noteworthy advantage: the Web technology stack has not yet achieved the level of performance we can attain with native code [9], thus mobile users will continue to seek native experiences in order to enjoy services of similar quality with the fixed Internet.

In addition, an increasing number of mobile users prefer to seek mobile content and services in their devices' application stores (e.g., the Apple Store). The so-called Mobile Application Store model is a new distribution paradigm, alternative to the dominant one of Mobile Web/Portal. The application creation and distribution paradigm underlying the idea is to grant higher openness and independence to third parties, following a "self-publishing model". From an offer perspective, the store's Web-oriented nature allows pairing the mature mobile content and service offer with Internet-based software applications [10].

4. Mobile (Native) Application

The customized application is written in Java and is developed in NetBeans 7.0 IDE using the Android Software Development Kit. It was developed to support all required news services (the handling of the news articles published on our news website) through mobile devices. Indicative functions are as follows:

• The retrieval and presentation of articles' categories.
• The retrieval and presentation of the ten most recent articles.

• The retrieval and presentation of the ten most recent articles per category.
• The publishing and deletion of an article.

It exploits the server-side XML-RPC methods and uses the client-side XML-RPC library for the Android platform.

4.1 Android Activities

The main services that the customized application may provide are listed below:

```
<activity android:name=".ViewArticle"></activity>
<activity android:name=".ListArticles"></activity>
<activity android:name=".NewArticle"></activity>
<activity android:name=".DeleteArticle"></activity>
<activity android:name=".ListCategories"></activity>
<activity android:name=".LatestArticles"></activity>
```

Let us provide at this point a few comments on indicative code which uses XML-RPC methods for retrieving a list of the most recent articles (Latest Articles activity):

```
public void onCreate(Bundle savedInstanceState)
 {  super.onCreate(savedInstanceState);
```

1. The first step is to create a client for the XML-RPC service that runs on the Joomla! server:

```
uri = URI.create("http://www.kasnakisg.gr/xmlrpc/index.php");
client = new XMLRPCClient(uri);
```

2. The second step is to create the appropriate user interface (UI):

```
setContentView(R.layout.main);
katigoria=this.getIntent().getExtras().getString("kat");
int kat=Integer.parseInt(katigoria);
errorDrawable =
    getResources().getDrawable(R.drawable.error);
errorDrawable.setBounds(0, 0,
    errorDrawable.getIntrinsicWidth(),
    errorDrawable.getIntrinsicHeight());
status = (TextView) findViewById(R.id.status);
tests = (ListView) findViewById(R.id.tests);
final ArrayAdapter<String> adapter = new TestAdapter(this,
    R.layout.test, R.id.title);
```

3. Remotely calling the getRecentPosts:

```
XMLRPCMethod method = new
    XMLRPCMethod("metaWeblog. getRecentPosts",
    new XMLRPCMethodCallback() {
        public void callFinished(Object result) {
```

4. In this step we receive as response an array of objects. The number of rows is examined and in case it exceeds

ten (10), the array is limited to the first ten rows only. If the number of rows is null, the user receives an appropriate message:

```
Object[] arr=(Object[]) result;
int i;
if  (arr.length>=10) numart1=10;
        else  numart1=arr.length;
```

5. The mobile application then decodes the structure into pairs of fields in the form of "Fields name"-"Fields price":

```
for(i=0;i<numart1;i++) {
    Map<String, Object> map = (Map<String, Object>) arr[i];
```

6. Every row of the array is transformed into a new object for the UI list structure, the 'adapter'. At the same time, all the necessary fields are saved to ensure that the onClick procedure works flawlessly:

```
adapter.add(map.get("postid").toString()+" –
        "+map.get("title").toString());
    desc.add(map.get("description").toString());  } }  });
```

7. The parameters have to be placed into an array:

```
Object[] params = { "",user, pass,10, kat,   };
```

8. The call is achieved through the method.call procedure of XML-RPC:

```
method.call(params);
```

9. The list object 'adapter' fills the test list container, as well as the testListener functionality.

```
tests.setAdapter(adapter);
tests.setOnItemClickListener(testListener);  }
```

10. When onClick is activated, a specific activity class is called with the appropriate parameter input:

```
OnItemClickListener testListener = new
  OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View
      view, int position, long id) {
        Bundle bundle = new Bundle();
```

11. The parameter test receives a string value that is the header for the html coding and the article's short description:

```
Bundle.putString("test", header+desc.get(position).toString());
```

12. The function ViewArticle is then called; it retrieves the content of the article.

```
startActivity(new Intent(view.getContext(),
    ViewArticle.class).putExtras(bundle));  }  };
```

13. Implementation of the list object. The list returns an integer that corresponds to the line's number:

```
class TestAdapter extends ArrayAdapter<String> {
    private LayoutInflater layouter;
    private int layoutId;
    public TestAdapter(Context context, int layoutId,
        int textId) {
        super(context, layoutId, textId);
        this.layoutId = layoutId;
        layouter = LayoutInflater.from(ListArticles.this);
}
@Override
public View getView(int position, View convertView,
    ViewGroup parent) {
        View view = layouter.inflate(layoutId, null);
        TextView title = (TextView)
                view.findViewById(R.id.title);
        TextView params = (TextView)
                view.findViewById(R.id.params);
    String string = getItem(position);
    String[] arr = string.split(";");
    title.setText(arr[0]);
    if (arr.length == 2) {
            params.setText(arr[1]);
    } else {
            params.setVisibility(View.GONE);
    }
    return view;  } }
```

14. As shown in the code below (part of the onCreate method of ViewArticle activity), the content of the article is being projected through a mini browser (WebView type):

```
 ...
browser=(WebView)findViewById(R.id.webkit);
strUrl=this.getIntent().getExtras().getString("test");
browser.loadData(strUrl,"text/html", "utf-8");    }
```

15. In the XMLRPCMethod, additions were made in order to handle errors and give appropriate messages when the remote method is being executed.

```
interface XMLRPCMethodCallback {
                void callFinished(Object result); }
class XMLRPCMethod extends Thread {
    private String method;
    private Object[] params;
    private Handler handler;
    private XMLRPCMethodCallback callBack;
    public XMLRPCMethod(String method,
        XMLRPCMethodCallback callBack) {
            this.method = method;
            this.callBack = callBack;
            handler = new Handler();        }
        public void call()     {
                call(null); }
```

```
            public void call(Object[] params) {
                status.setTextColor(0xff80ff80);
                status.setError(null);
                status.setText("Calling host " +
                        uri.getHost());
                this.params = params;
                start();
            }
        @Override
        public void run() {
        try {
            final long t0 = System.currentTimeMillis();
            final Object result = client.callEx(method, params);
            final long t1 = System.currentTimeMillis();
            handler.post(new Runnable()      {
                public void run() {
                    status.setText("XML-RPC call took " + (t1-t0) +
                        "ms");
                    callBack.callFinished(result);  }
                                });
        } catch (final XMLRPCFault e) {
            handler.post(new Runnable() {
                public void run() {
                    status.setTextColor(0xffff8080);
                    status.setError("", errorDrawable);
                    status.setText("No available articles");
                    Log.d("Test", "error", e);
                            }
                    });
        } catch (final XMLRPCException e) {
            handler.post(new Runnable() {
                public void run() {
                    status.setTextColor(0xffff8080);
                    status.setError("", errorDrawable);
                    Throwable couse = e.getCause();
                    if (couse instanceof
                        HttpHostConnectException) {
                        status.setText("Cannot connect to " +
                            uri.getHost());
                                    }
                    else {
                     status.setText("Articles Not Found" +
                            e.getMessage());
                    }
                    Log.d("Test", "error", e);
                        }
                    });
                }
            }
    }      }
```

4.2 On the Server Side

On the server side, a request to the data base is executed and results are being received:

```
function getRecentPosts($blogid, $username, $password,
    $numposts) {
    global $xmlrpcerruser, $xmlrpcI4, $xmlrpcInt,
    $xmlrpcBoolean, $xmlrpcDouble, $xmlrpcString,
    $xmlrpcDateTime, $xmlrpcBase64, $xmlrpcArray,
    $xmlrpcStruct, $xmlrpcValue;
```

The following code is for identifying the user:

```
If (!plgXMLRPCmetaWeblogHelper::authenticateUser
    ($username, $password))
return new xmlrpcresp(0, $xmlrpcerruser+1, "Login Failed");
$user =& JUser::getInstance($username);
$aid  = plgXMLRPCmetaWeblogHelper::getUserAid($user->id);
$plugin =& JPluginHelper::getPlugin('xmlrpc','metaweblog');
$params = new JParameter( $plugin->params );
```

A data base object is being created:

```
    $db =& JFactory::getDBO();
```

A query is being executed:

```
$query = 'SELECT c.id, c.title, c.alias, c.created_by, c.introtext,
        c.created, c.state'
    .' FROM #__content AS c'
    .' INNER JOIN #__sections AS s ON c.sectionid = s.id'
    .' INNER JOIN #__categories AS cc ON c.catid = cc.id'
    .' WHERE s.published = 1 AND cc.published = 1'
    .' AND s.access <= '.$aid .' AND cc.access <= '.$aid.' AND
        c.access <= '.$aid .' AND c.state >= 0'
    .' ORDER BY c.created DESC';
$db->setQuery($query, 0, $numposts);
$items = $db->loadObjectList();
```

A check for results is being performed and if results are found they undergo a proper serialization before transfer.

```
if  (!$items) return new xmlrpcresp(0, $xmlrpcerruser+1, 'No
    posts available, or an error has occurred.' );
require_once (JPATH_SITE.DS.'components'.DS.'com_content'
    .DS.'helpers'.DS.'route.php');
$structArray = array();
foreach ($items as $item)    {
  $dateCreated=& new JDate($item->created);
  $articleLink= JURI::root()
    .(ContentHelperRoute::getArticleRoute($item->id, $item-
    >catid, $item->sectionid));
  $structArray[] = new xmlrpcval(array(
    //'dateCreated' => new xmlrpcval($dateCreated-
    >toISO8601(), 'dateTime.iso8601'),
    'title'           => new xmlrpcval($item->title),
    'description'    => new xmlrpcval($item->introtext),
    'userid'          => new xmlrpcval($item->created_by),
    'postid'          => new xmlrpcval($item->id),
    'link'            => new xmlrpcval($articleLink),
    'permaLink'      => new xmlrpcval($articleLink)
    ), $xmlrpcStruct);  }
return new xmlrpcresp(new xmlrpcval( $structArray ,
    $xmlrpcArray));  }
```

The return is an array of articles having selected fields. All activities of the Android application follow the model described above.

4.3 Use of RSS/Atom Feeds

We have to underline that the mobile application is capable of using RSS/Atom feeds: as we will describe in the next sections, significant effort has been made to build a news website with advanced characteristics for feeds. Consequently, the mobile application takes advantage of this fact, so that the content will be presented to the mobile user more concisely. The user can choose from a list in the category that she is interested in and only titles of articles that belong in that category will be presented. After selecting a title, the whole article can be projected to the user. Figures 1 and 2 depict the relative class and sequential diagrams, correspondingly.

5. Mobile News Website

5.1 Joomla! Extension/Component Development

Mobile Joomla! is a powerful open source component regarding mobile device-oriented adaptations for Joomla! websites. It is available in the BackEnd Joomla! environment and it allows website administrators to choose their preferred way of assigning templates (using standardized templates, customizing the predetermined templates or supporting TERA WURFL mechanisms). Mobile Joomla! allows adding or removing menus, modules or other user interface components. Moreover, it favours the selective placement of components: it contains modules capable of modifying the export of the news website content and placing it in suitable template containers. For example, it is possible to only present the introduction of an article and hide information such as the name of the author or the date on which the article was published.

The main Joomla! tables (related to the news-oriented website) are the Content table, the Categories table and the Sections table (Figures 3, 4, 5). Content (or article) is the basic entity that belongs to a category which in turn belongs to a section. One section may contain many categories and one category may contain many articles.

Components are the main functional units of Joomla!; they can be seen as mini-applications. They are a kind of Joomla! extension. To handle consistently external content sources, such as feeds and emails, a specific Backend Joomla! component (Rsstoa) was built. The Joomla! code has been designed for extensibility [12], thus the process of developing a component should be according to the Model-View-Controller (MVC) software design pattern:

• The model is the part of the component that encapsulates the application's data. It will often provide routines to manage and manipulate this data in a meaningful way, in addition to routines that retrieve the data from the model.

• The view is the part of the component that is used to render the data from the model in a manner that is suitable for interaction. For a Web-based application, the view would generally be an HTML page that is returned to the user. The view pulls data from the model (which is passed to it from the controller) and feeds the data into a template which is populated and presented to the user.

• The controller is responsible for responding to user actions. In the case of a Web-based application, a user action is (generally) a page request. The controller will determine what request is being made by the user and respond appropriately by triggering the model to manipulate the data appropriately and passing the model into the view. The controller does not display the data in the model, it only triggers methods in the model which modify the data, and then pass the model into the view which displays the data.
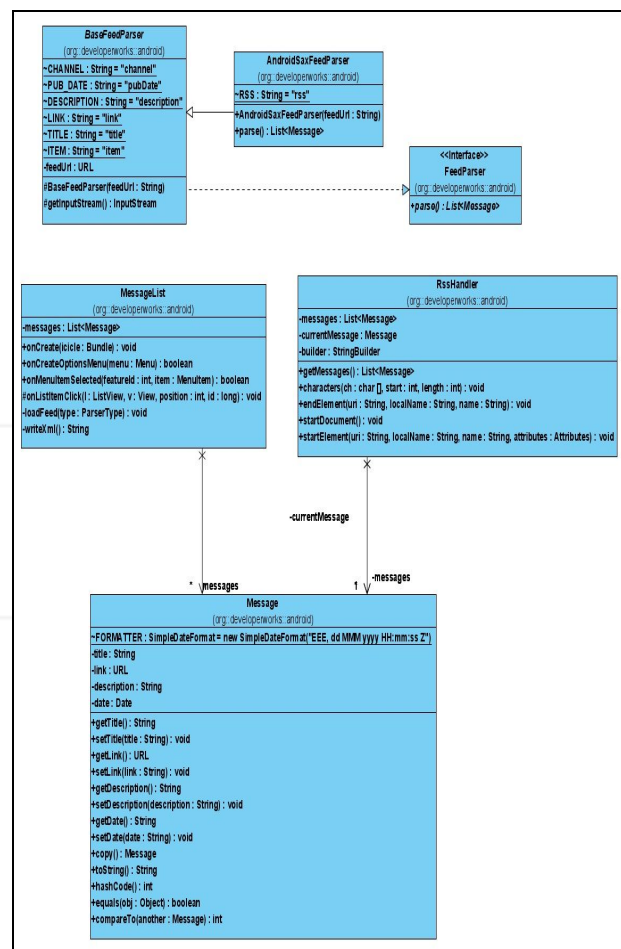

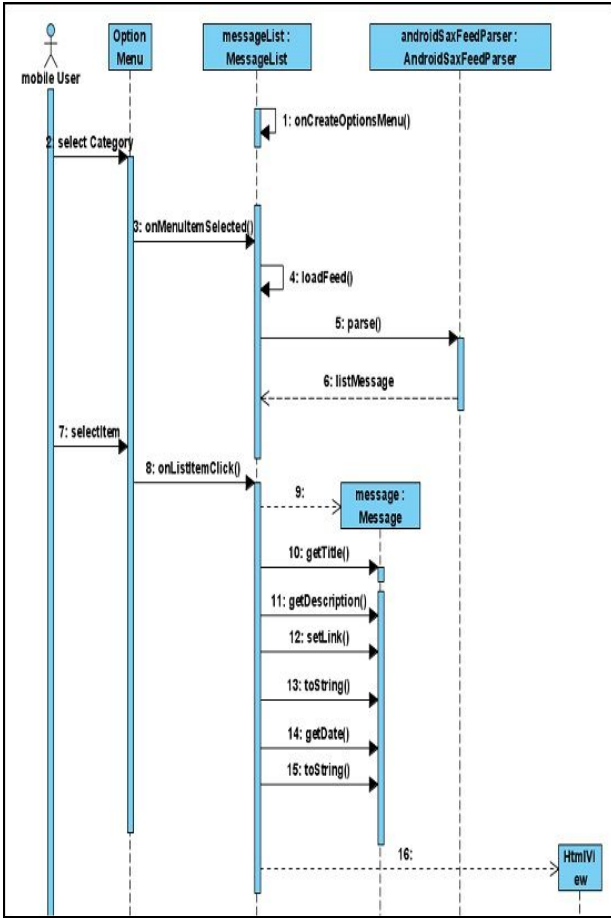**Figure 1.** Feeds via the Mobile Application - Class Diagram

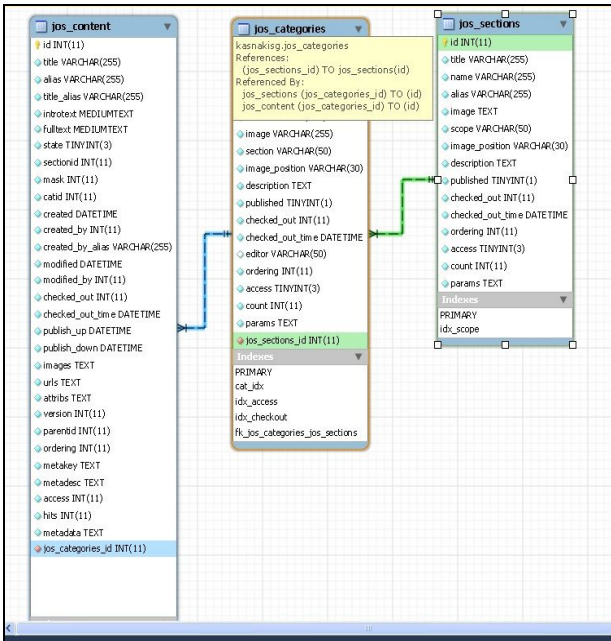**Figure 2.** Feeds via the Mobile Application: Sequential Diagram

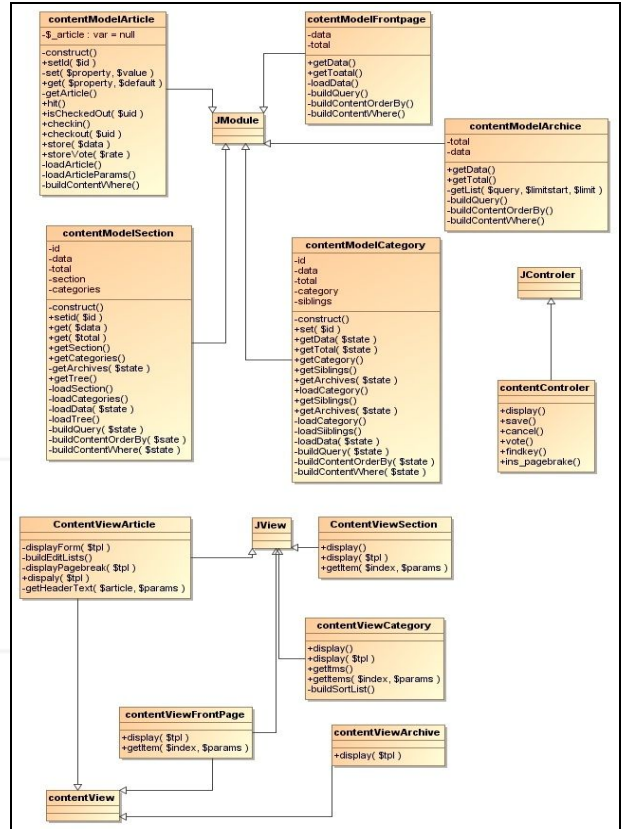**Figure 3.** The Section-Category-Article Structure
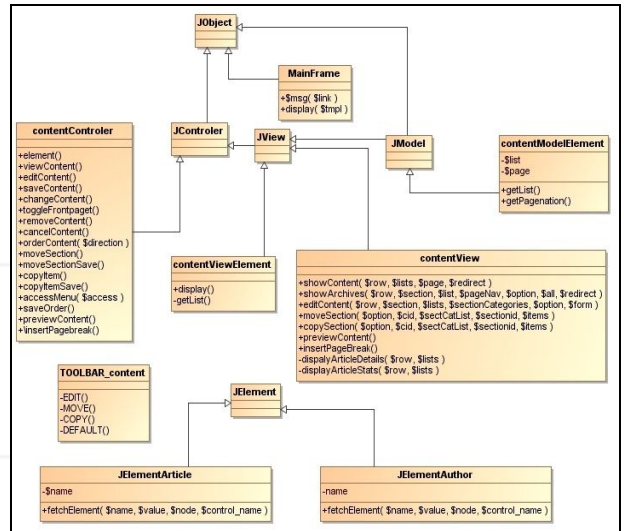

Figure 4. Frontpage Class Diagram

Figure 5. Backend Class Diagram

Figure 6 depicts the directory structure of the Rsstoa component, and Figure 7 presents its robustness diagram (to express the entity elements, the interface/boundary elements and the process elements between them).
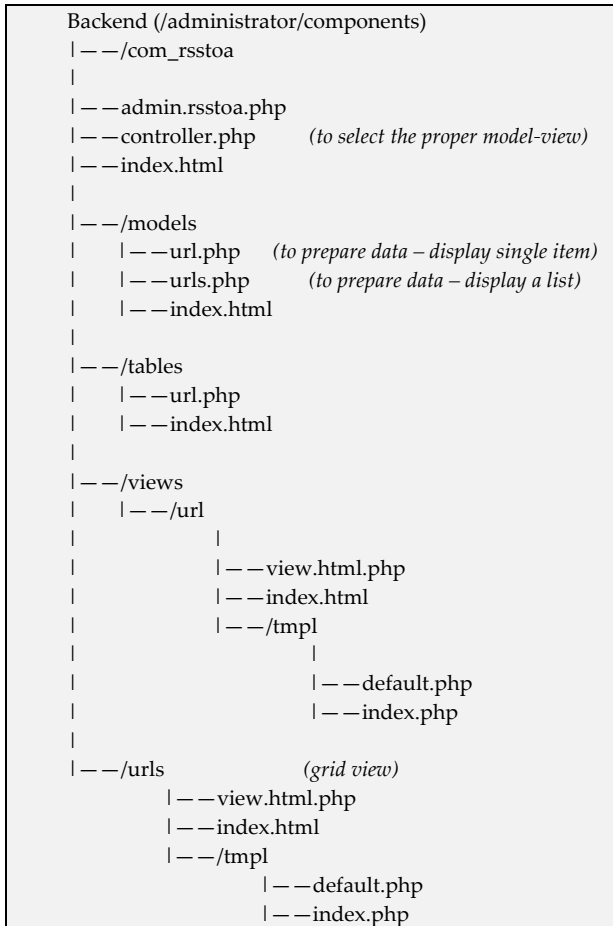
```
Backend (/administrator/components)
|——/com_rsstoa
|
|——admin.rsstoa.php
|——controller.php        (to select the proper model-view)
|——index.html
|
|——/models
|    |——url.php      (to prepare data – display single item)
|    |——urls.php        (to prepare data – display a list)
|    |——index.html
|
|——/tables
|    |——url.php
|    |——index.html
|
|——/views
|    |——/url
|    |          |
|               |——view.html.php
|               |——index.html
|               |——/tmpl
|               |          |
|                          |——default.php
|                          |——index.php
|
|——/urls                    (grid view)
           |——view.html.php
           |——index.html
           |——/tmpl
                      |——default.php
                      |——index.php
```

Figure 6. The Joomla! Directory Structure of Rsstoa Component
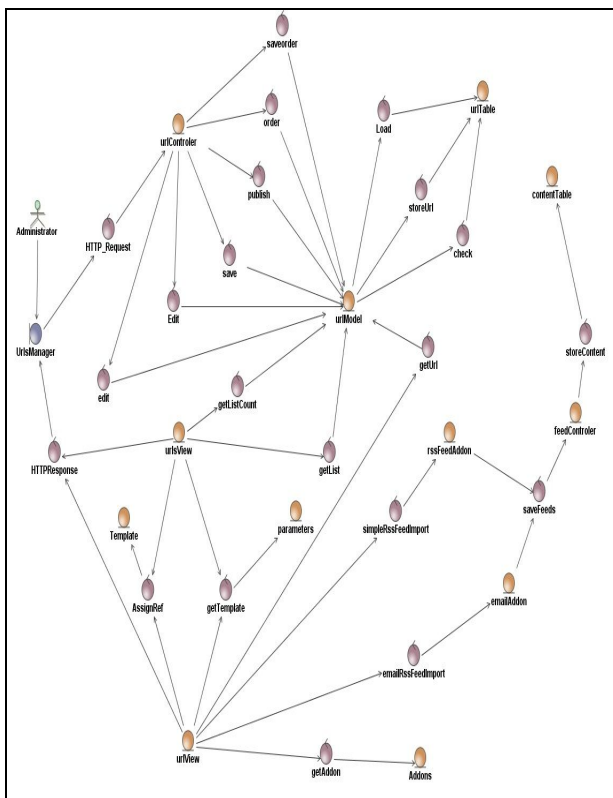
Figure 7. Rsstoa Component: Robustness Diagram

5.2 Publishing Content through RSS/Atom Feeds

RSS/Atom feeds for a news website could be used for indexing in order to create news articles from various sites and sources clustered together in one website. The first major modification/improvement on Joomla!'s default functionality was storing feeds in the website's database. Subsequently, the stored articles might be filtered by keywords or even other feeds' metadata (such as date or origin). Joomla! has a function named Jfactory::getXMLparser which accepts an RSS or ATOM feed as input and returns a PHP-Based RSS and Atom Feed Framework object, namely the SimplePic. An indicative approach for storing RSS feeds in the Joomla! Content table is by using its fields as follows:

```
$contenttable->title=$title;
$contenttable->permalink=$rssurl->section_id;
$contenttable->catid=$rssurl->cat_id;
$text=$item->get_content();
$contenttable->created=$item->get_date('Y-m-d H:i:s');
$contenttable->publish_up=$item->get_date('Y-m-d H:i:s');
```

We may preserve Content Joomla! table using a secondary table (feedtable) in which information regarding the author, the hyperlink of the source and other information will be stored.

```
$premalink=$item->get_permalink();
$feedtable->sourcelink=$premalink;
```

The capability to have multiple feeds, which can be enhanced with filters, could work as a mechanism for article indexing from selected sources. On the other hand, indexing could also be achieved through social network applications such as the 'Yahoo pipes' mashup. An author may adjust a mashup for her own needs and interests and then send those feeds to the news website in an RSS or JSON format. In any case, if the advantages of permanent storage are required, these data have to be stored in the site's database.

5.3 Publishing Content via Email

The second indicative functionality that we investigated and developed through Rsstoa component was related to emails. Users with rights to publish articles (both members and authors) may in this way send an article via email and that article will be automatically published in the appropriate section. All that is needed is the author's approval.

The necessary library for the management of the emails is provided directly by PHP. The library is called IMAP and it provides the functions that allow the handling of the email structure. An email parser is actually developed; it is capable of splitting the email into discrete pieces (title,

main body and attachments). These pieces may then be used for the creation of the article just like in the case of the RSS feeds. Consequently, this integrated approach of handling both RSS/Atoms and emails leads to the formalization of certain user cases for managing URL feeds, as Figure 8 illustrates.
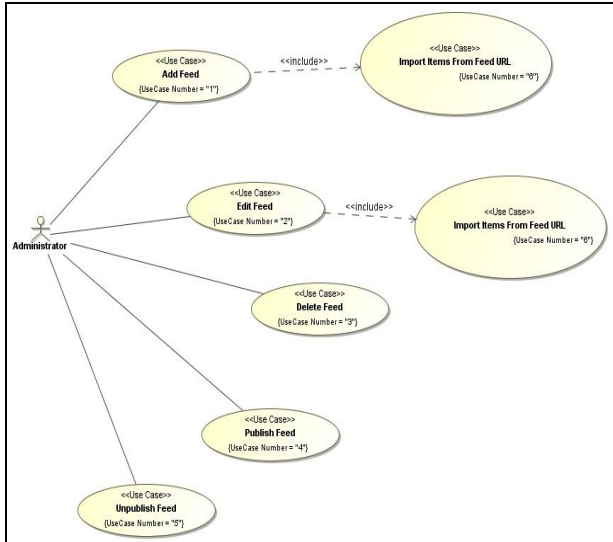

Figure 8. User Cases for Managing RSS/Atom Feeds and Emails

The storage of the URL of a certain feed or of a certain email triggers (triggerEvent) the processes of importing RSS items (SimpleRssFeedImport) or emails (emailRssFeedImport) in the Content table and simultaneously the update in the secondary table (Feedtable) of the additional fields that cannot be saved in the basic table (Figure 9).
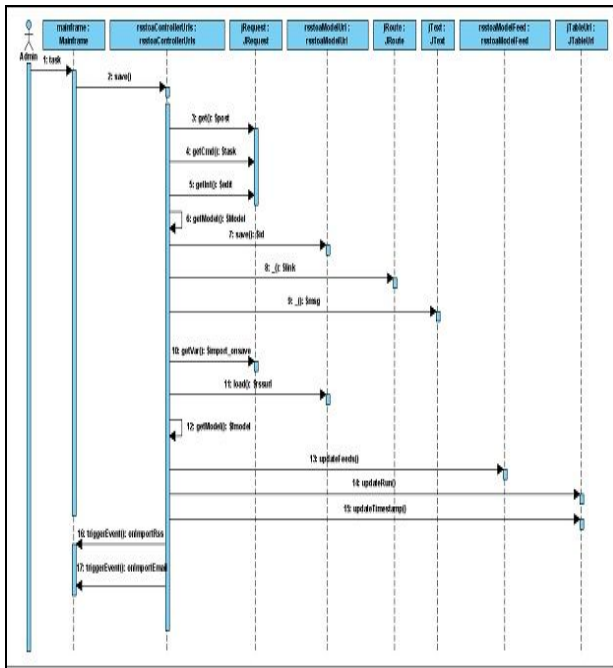

Figure 9. Save and Edit Feed/Email: Sequential Diagram

According to the MVC pattern, Rsstoa component's model class had to be responsible for executing the required database queries (Figure 10).
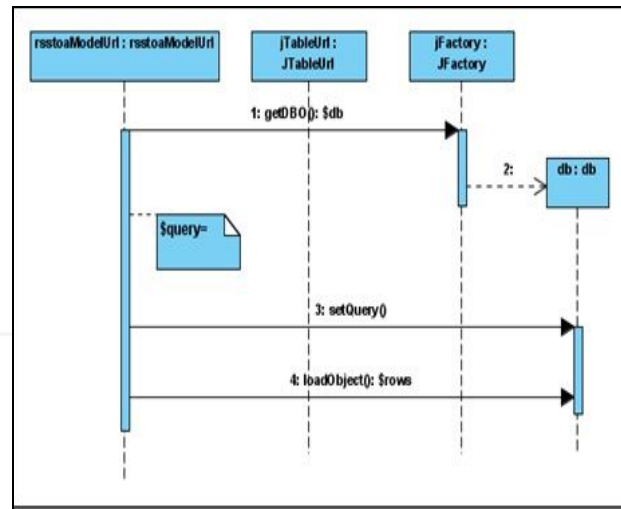

Figure 10. getList(): Sequential Diagram

An important function which we had to implement was the getAttachments() function. Its job was to extract the email body, as well as all potential attachments. Figure 11 depicts its sequential diagram. The decomposition of the emails is implemented via the function fetchstructure(), based on the returned objects' values of Table 1.
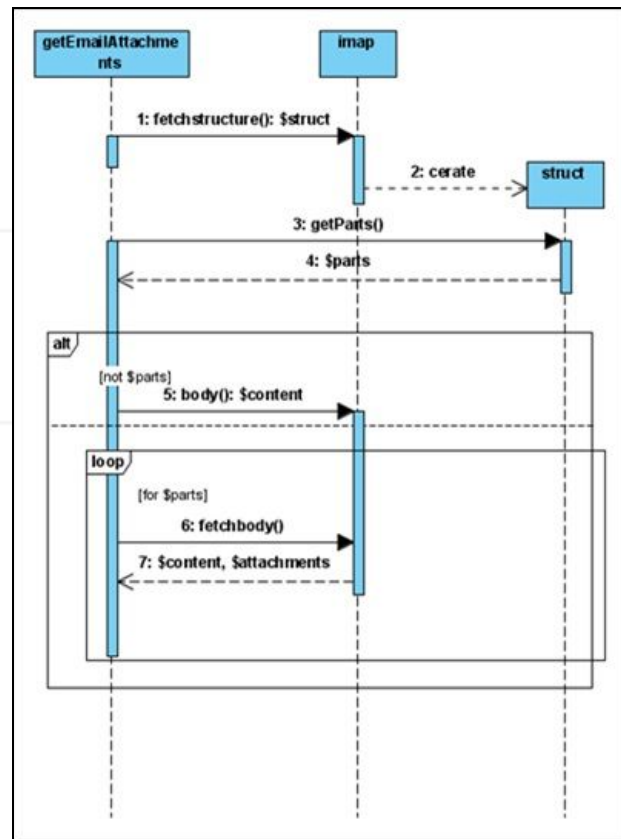

Figure 11. getAttachments(): Sequential Diagram

|	
type	Primary body type
encoding	Body transfer encoding
ifsubtype	TRUE if there is a subtype string
subtype	MIME subtype
ifdescription	TRUE if there is a description string
description	Content description string
ifid	TRUE if there is an identification string
id	Identification string
lines	Number of lines
bytes	Number of bytes
ifdisposition	TRUE if there is a disposition string
disposition	Disposition string
ifdparameters	TRUE if the dparameters array exists
dparameters	An array of objects where each object has an "attribute" and a "value" property corresponding to the parameters on the Content-disposition MIME header.
ifparameters	TRUE if the parameters array exists
parameters	An array of objects where each object has an "attribute" and a "value" property.
parts	An array of objects identical in structure to the top-level object, each of which corresponds to a MIME body part.

Table 1. Returned Objects for imap_fetchstructure()

The saveEmail ($mbox, $mid, $rssurl) function updates Content and feedtable tables. Function's inputs are the mailbox, the message id and the Rsstoa object which contains parameters (stored in the corresponding xml file) concerning important settings such as the category in which the mail should be imported, the reference of the source, the date of publication, if the email should be published on the front page of the website (Figure 12).
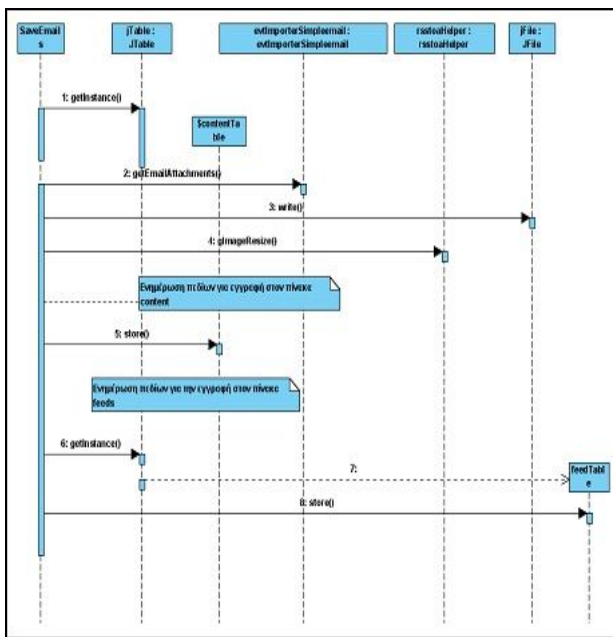
Figure 12. saveEmail(): Sequential Diagram

In Joomla! components may have additional libraries, namely Helpers, that contain useful functions/tools. In Figure 13, imageResize() function is depicted. It was created using php methods.
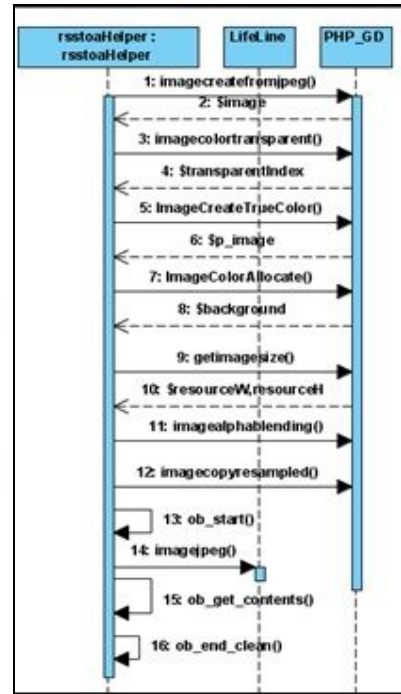
Figure 13. imageResize(): Sequential Diagram

Let us provide again at this point a few comments on indicative php code. Steps 1-2 for creating an image object from an image file:

```
$image= @imagecreatefromjpeg($file);
```

Steps 3-4 and 9 for storing the parameters (transparent, width, height) of the initial image temporarily:

```
$transparentIndex= imagecolortransparent( $image );
 list($currentWidth , $currentHeight) =
                    getimagesize($srcPath );
```

Steps 5-8 for creating colours:

```
$image_p= ImageCreateTrueColor($destWidth,
               $destHeight);
$background= ImageColorAllocate($image_p, 255, 255, 255 );
```

Steps 9-12 for allocating the appropriate temporary memory space to hold the image copy:

```
imagealphablending( $image_p , false );
imagecopyresampled($image_p ,$image, $targetX, $targetY,
          $sourceX, $sourceY, $destWidth , $destHeight,
          $currentWidth , $currentHeight);
```

Steps 13-14 refer to a very interesting process: the ob_start() function directs output to buffers. Until the

execution of the ob_end_clean() process, we may send all data (in our case the memory content/variable $image_p) to buffer:

```
imagejpeg($image_p, null, 80);
```

Finally, steps 15-16 for copying buffer content to a variable (image object) and emptying the buffer:

```
$output = ob_get_contents();
ob_end_clean();
```

In Joomla! the MVC pattern is implemented using three classes: JModel, JView and JController, as it is depicted in Figure 14. The sequential diagram of the Rsstoa component is presented in Figure 15.
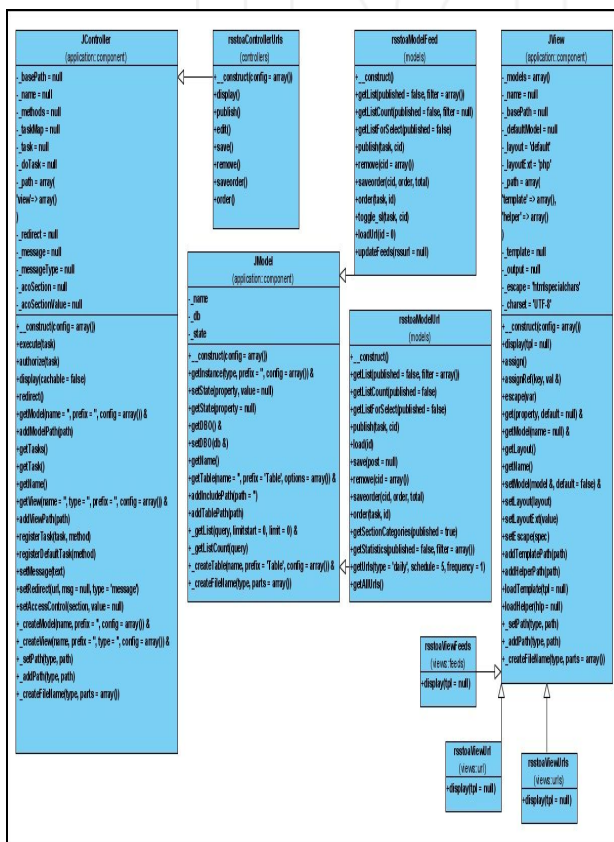

Figure 14. Rsstoa Component: Class Diagram

6. Conclusions and Results – Further Research

A series of trials and tests were performed to ensure that the website operates flawlessly:

- Articles were published using RSS feeds and regular emails.
- We allowed selected users to comment on the articles, to avoid future problems.
- Articles were shared in well-known social networking sites.
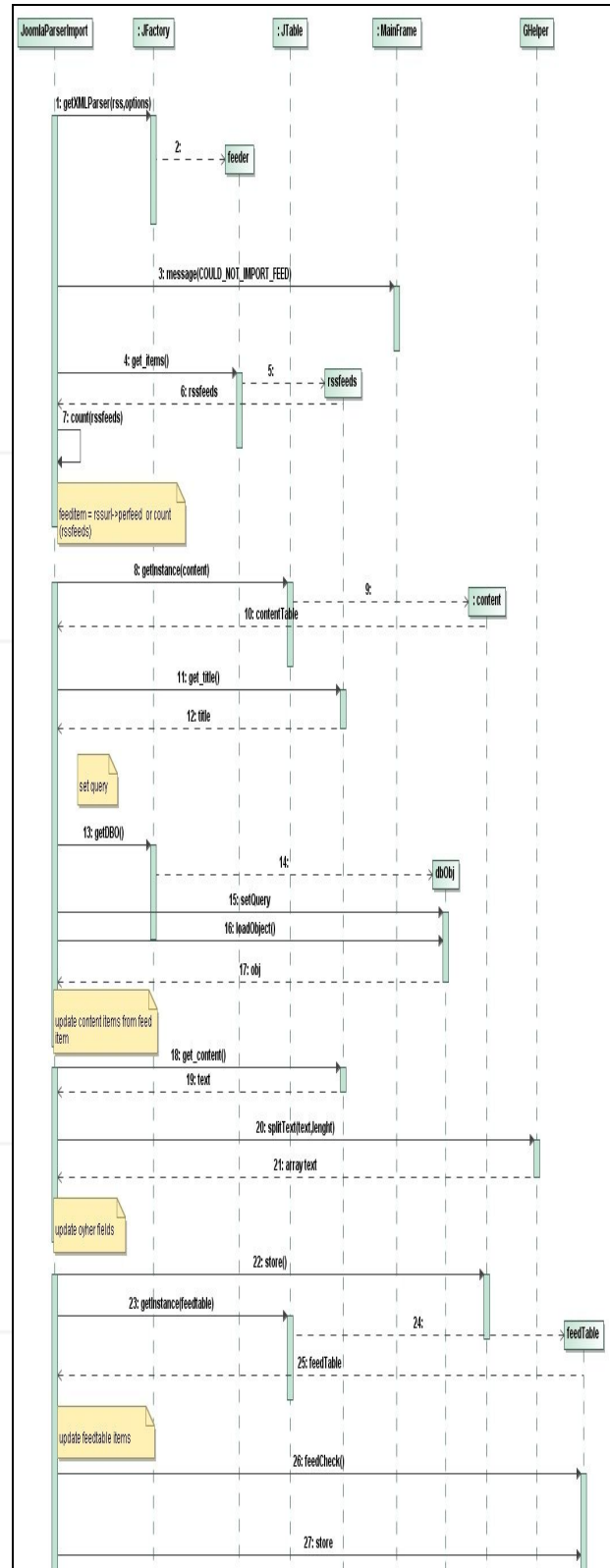- Articles were handled through the mobile application.

Figure 15. Rsstoa Component: Sequential Diagram

The existence of feeds allows users to be informed on various subjects faster and more completely. Nevertheless, the simple republishing of articles that were firstly published on other websites decreases the website's value as a source of information. Consequently, to increase the

business perspectives of news intermediary services more attention has to be given to organizing the selection, organization, hierarchical ranking and distribution of the news published by others. This new 'industry of access' to content produced externally may raise questions concerning the pluralism of news. Current research indeed argues that while it appears that the Internet has theoretically favoured the emergence of countless 'new' news channels, it seems not to have fostered as many new sources, so much as new gateways to access the same contents produced by a few companies [1].

It is worth mentioning at this point a significant business-related pitfall that a news infomediary service developer has to avoid: given the well-known heavy criticism for copyright infringement by the Google News service [12], our approach favours portal-oriented procedures: agreements/deals with news providers, similar to the ones a press company agrees with a news agency – a subscription to a newswire with an agreement to use a certain number of news items each day.

Further research is needed for the development of an additional component capable of handling relative personalization issues (such as recommending specific feeds or user comments, based on users' profiles and behaviours). Also, an interesting extension could be the development of supplementary mobile news services, by designing and implementing a mobile application that would enable more features like the geographical identification of the source of news feed, the sharing of news based on geographical criteria and the utilization of REST services that are offered in many social networking sites.

7. Acknowledgments

8. References

[1] Goyette-Côté M-O, Carbasse R, George E (2012) Converging Journalism, Journalism Studies, iFirst Article: 1-10.

[2] van Loon S (2012) The Power of Google: First Mover Advantage or Abuse of a Dominant Position? In A. Lopez-Tarruella (ed.), Google and the Law, Information Technology and Law Series 22: 9-36.

[3] Cortimiglia M, Ghezzi A, Renga F (2011) Social Applications: Revenue Models, Delivery Channels, and Critical Success Factors – An Exploratory Study and Evidence from the Spanish-Speaking Market. Journal of Theoretical and Applied E-commerce Research, Special Issue on Business Models for Mobile platforms – Vol. 6, Issue 2: 108-122.

[4] Georgiadis CK (2012) Design and Implementation of Mobile News Services: Supporting Social Networking Features. In Proceedings of the AIS-affiliated Eleventh International Conference on Mobile Business. Delft, pp. 101-112.

[5] Wittenbrink H (2005). RSS and ATOM: Understanding and Implementing Content Feeds and Syndication. Puckt Publishing. 256p.

[6] Georgiadis CK (2010) Developing Personalized Information Services for Mobile Commerce Location-Aware Applications. Int. J. on Advances in Internet Technology, 3 (3&4): 274-283.

[7] van Woelsen W, Casteleyn S, de Troyer O (2011) A Generic Approach for On-The-Fly Adding of Context-aware Features to Existing Websites. In Proceedings of the 22nd ACM Conference on Hypertext and hypermedia (HT '11), ACM, USA, New York. pp. 143-152.

[8] Zhang D, Lai J (2011) Can Convenience and Effectiveness Converge in Mobile Web? A Critique of the State-of-the-Art Adaptation Techniques for Web Navigation on Mobile Handheld Devices. Int. J. of Human-Computer Interaction, 27:12: 1133-1160.

[9] Charland A, Leroux B (2011) Mobile application development: Web vs. Native. Commun. ACM 54, 5: 49-53.

[10] Ghezzi A, Balocco R, Rangone A (2010) How a New Distribution Paradigm Changes the Core Resources, Competences and Capabilities Endowment: The Case of Mobile Application Stores. In Proceedings of Ninth International Conference on Mobile Business/ Ninth Global Mobility Roundtable, Athens. pp. 33-42.

[11] Kennard J, Lanham C (2010). Mastering Joomla 1.5! Extension and Framework Development Second Edition: The Professional Guide to Programming Joomla!. Puckt Publishing. 562p.

[12] Laurent P (2011) Copiepresse SCRL & alii v. Google Inc. - In its decision of 5 May 2011, the Brussels Court of Appeal confirms the prohibitory injunction order banning Google News and Google's "in cache" function. Computer Law & Security Review 27: 542-545.

[13] Ghezzi A, F Renga, R Balocco (2009) A Technology Classification Model for Mobile Content and Service Delivery Platforms. In: Joaquim Filipe, José Cordeiro (Eds.). Enterprise Information Systems – Lecture Notes in Business Information Processing. Verlag: Springer. 24(3): 600-614. doi: 10.1007/978-3-642-01347-8_50.