CHAPTER 4

E-kones Experience Architecture

This study examines the application support for the user that have bought a tourism package from the E-kones service, and now he/she experiences that package. That's why the application named "E-kones Experience".

This study took in account the specific characteristics of a portable device (small screen size, limited memory) and the effort concentrated into an application that is more to desktop standards rather in mobile standards. Therefore were developed interface components that can simulate that behavior. The result was a hybrid application which in some parts utilizes good techniques from mobile standards (like central menu) and in some others tries to offer components that are common in classical desktop interfaces (combobox, popup, menus etc).

For the application needs were developed the next components

- XML Parser (Reader)
- Custom Components (MenuItem, ComboBox, PopUp, RateItem,Container)
- Calendar
- Map

The application has been build around two major components The Calendar and the Map, with the rest to provide support to the major components. We could say that the Calendar object has the "Time" perspective for the user in the package, while the "Map" object has the spatial. In the next diagram (image 4.1) it is presented the architecture of the final prototype version.

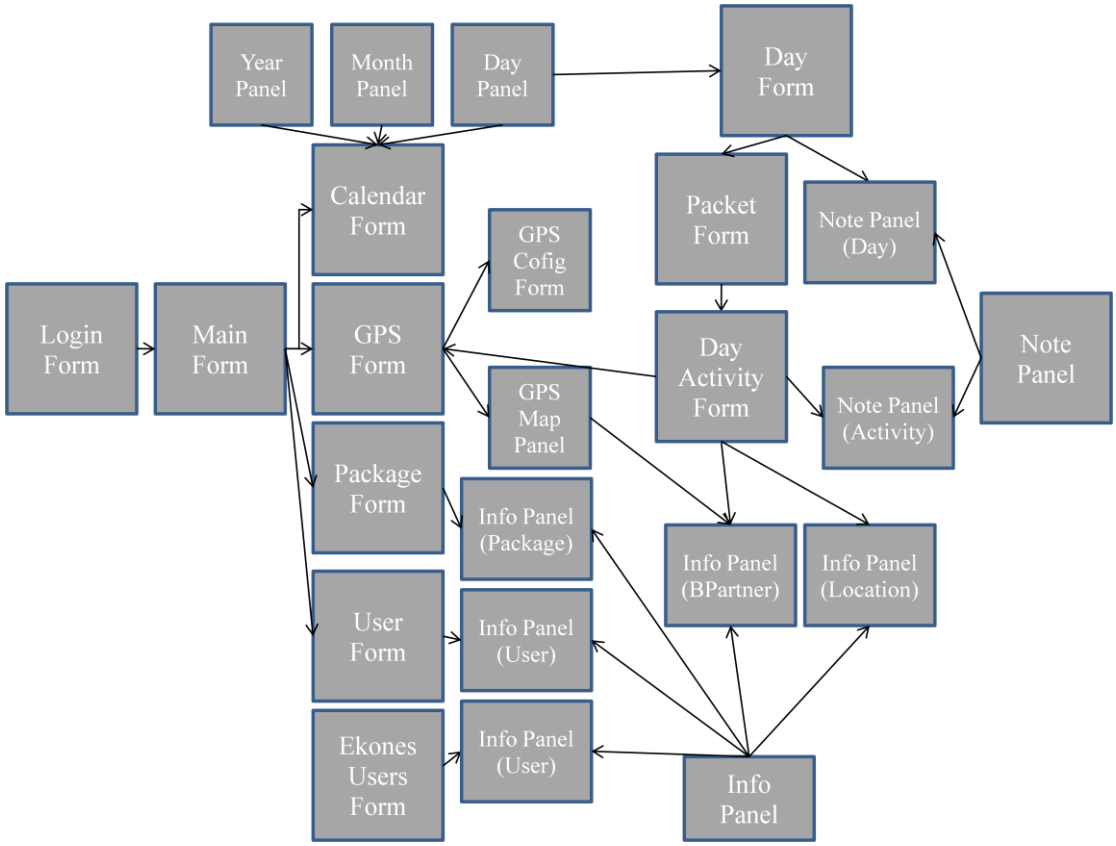


image 4.1 Application Architecture in the final prototype version.

Application Components

- **IstlContainer**
- **IstlLayout**
- **IstlComboBox**
- **IstlMenuItem**
- **IstlLabel**
- **IstlPopUp**
- **IstlRateItem**

IstlContainer

istlContainer Component extends the LWUIT container. The only difference with LWUIT Container is that the istlContainer can have ActionListeners. IstlContainer was developed in order to receive "pen –touch screen" events, which are vital in order to send and receive the pen position on the screen, it has two more methods.

Return type	Method name	Usage
int	getXcor()	Returns the X screen coordinate
int	getYcor()	Returns the Y screen coordinate

The istlContainer is used by the Application mainly in istlMap component.

IstlLayout

The layout manager is responsible for positioning other components in the user's desired position, with the use of 2 methods setX() and setY(). The istlLayout layout manager is widely used by the application for putting most of the components on their containers. Mainly used by the istlMap Componet where permits the positioning of the markers in the desired user's click position.

IstlLabel

The istlLabel is a hybrid component that came as a result by putting together elements from the Button component and from the label component, and extends the LWUIT label.

Return type	Method name	Usage
int	getXcor()	Returns the X screen coordinate
int	getYcor()	Returns the Y screen coordinate
void	setLabelid(int lid)	Assigns users id value
int	getLabelId()	Returns user's id value

IstlComboBox

This component extends the LWUIT Container and it is implemented with the usage of two already existed components. Diagrams (image 4.2 and 4.3) present its architecture.

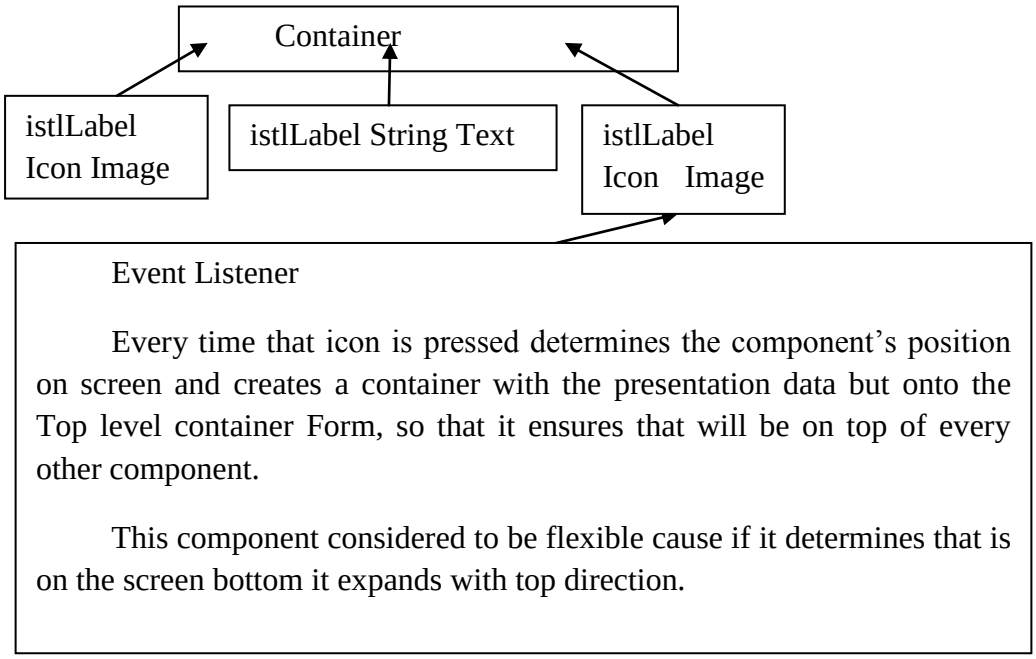


Image 4.2 General architecture of istlComboBox component.

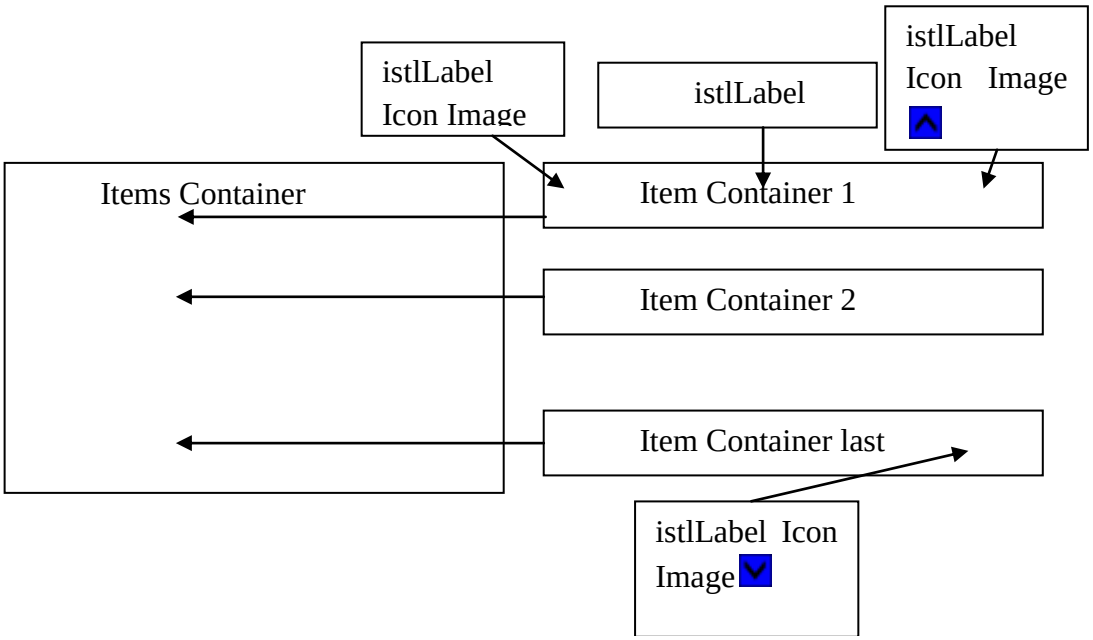


Image 4.3 General architecture of istlComboBox component in deployed state

istlComboBox component, in first diagram (image 4.2) it's the component itself, in the second diagram (image 4.3) is the architecture of the deployed part that appears when the arrow button is pressed .

In this version the total amount of elements that are appearing on screen at a time is locked at 4, which isn't mutable with some method. Despite that there is a scroll bar that enables when detects more than 4 components.

In order to construct an istlcombobx a model class is used, which holds the object of every selection, this class recognizes automatically if the selection is image,

text or both. Providing the ability of having selections of different types in the same istlcombobox

usage

```
Vector mpt = new Vector();
istlComboModelItem cmbi = new
istlComboModelItem(st.getMapTypeHybridIco(),"Hybrid");
mpt.addElement(cmbi);
cmbi = new istlComboModelItem(st.getMapTypeMobileIco(),"Mobile");
mpt.addElement(cmbi);
cmbi = new
istlComboModelItem(st.getMapTypeSatteliteIco(),"Satellite");
mpt.addElement(cmbi);
cmbi = new istlComboModelItem(st.getMapTypeTerrainIco(),"Terrain");
mpt.addElement(cmbi);

maptypes = new istlComboBox(mpt);
maptypes.setDefaultIcon(st.getMapTypeIco());
```

The code above creates the istlcombobx for the map type selection (image 4.4).

Εικόνα 4.4
istlComboBox

A new vector is constructed with the name `mpt` which will contains items of `istlComboModelItem` type, each item of those will be a selection for the istlcombobox in this case each selection will consist of an image and a title. After an object of istlcomboBox is created with `maptypes` name and with argument that Vector (mpt) (`new istlComboBox(mpt);`), also with the method (`.setDefaultIcon(st.getMapTypeIco());`) an image is given as the default.

After, with the usage of an ActionListener on a button we can take the selected value of the maptypes stlComboBox that we previously create, as appears in the following part of code.

```
apply.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
maptypes.getSelectedItem();
            }
        });
```

This part of code assigns an ActionListener on Apply button of the Map Configuration form (will be examined later). Each time that the Apply button is pressed the istlComboBox return the selected value by using the method `.getSelectedItem();`

IstlMenuItem

This component extends the LWUIT container and it is build with the use of two existing components. Its architecture is a modified version of istlComboBox, but having similar characteristics and properties.

A difference with istlComboBox is in the model class. In order to construct an IstlMenuItem it is used a model class, that contains the item of each choice, this class can automatically determines if the selection item is image, text or both, providing the ability of having an istlMenuItem with a combination of selections in the same istlMenuItem, in addition in order to support a second depth level it can accept another istlMenuItem as argument. In order this to happen the istlMenuItem that should go as a 2nd level must build first, and then must be added to the Vector as element, finally the Vector with the elements must go as an argument to the 1st level menuitem. Such item is the istlMenuItem that exists in istlMap component (image 4.5). Below we examine a code sample of how this item created.

image4.5 istlMenuTiem

Usage

```
Vector mpt = new Vector();
istlMenuModelItem cmbi = new istlMenuModelItem("Hide");
mpt.addElement(cmbi);
cmbi = new istlMenuModelItem(st.getNotationAddIco(),"Add");
mpt.addElement(cmbi);
cmbi = new istlMenuModelItem(st.getNotationMoveIco(),"Move");
mpt.addElement(cmbi);
cmbi = new istlMenuModelItem(st.getNotationRemoveIco(),"Remove");
mpt.addElement(cmbi);
sub_menu = new istlMenuItem("Notations",mpt);
sub_menu.setDefaultIcon(st.getNotationPropertiesIco());

sub_menu.getMenuitemTitle(0).addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
istlLabel current = (istlLabel) evt.getSource();
sub_menu.getMenuItems().setVisible(false);
menu.getMenuItems().setVisible(false);
t.getComponentForm().removeComponent(sub_menu.getMenuItems());
t.getComponentForm().removeComponent(menu.getMenuItems());
for(int k=0;k<t.getComponentForm().getComponentCount();k++)
{
t.getComponentForm().getComponentAt(k).repaint();
t.getComponentForm().getComponentAt(k).refreshTheme();
}
}
});
```

The construction technique is similar with the istlComboBox component, unfortunately due to a problem which still remains unsolved (due to lack of time) it is necessary that the programmers should take care of disappearing the specific part of each selection (like in the previous code sample).

```
Vector zmt = new Vector();
istlMenuModelItem cmbr;
cmbr = new istlMenuModelItem("Gps Config");
zmt.addElement(cmbr);
cmbr = new istlMenuModelItem(sub_menu);
```

```
cmbr = new istlMenuModelItem("Exit");
zmt.addElement(cmbr);
menu = new istlMenuItem("Options",zmt);
```

After all the menus are added to the central menu named menu.

IstlPopUp

It is a simplified version of the istlMenuItem with one level depth. It can be constructed with the same way as an istlMenuItem. Image 4.6 shows an istlPopUp with selections for Bussiness Partners.

image 4.6
istlPopUp

IstlRateItem

This component in contrast with all the previous doesn't try to simulate behavior from desktop computers interfaces, but it is a total new component by itself, therefore there was effort in order this component to have maximum flexibility and portability.

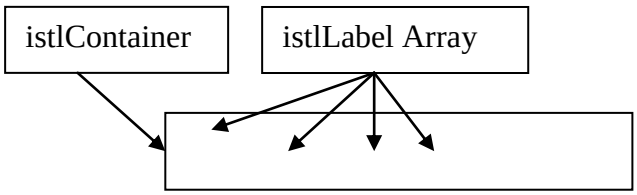


image 4.7 istlRateItem Architecture

Diagram of image 4.7 shows the basic architecture of the istlRate Component. The logic is simple, an istlContainer is made with symbols that represents rating. There is a constructor that needs a number of rate items as argument (example 5 or 8), the default rate symbol is the star , but also a variety of methods are provided in order to add custom symbols, distances between symbols, and orientation. The istlRate component utilizes methods for identifying characteristics like screen width and auto fit its rate symbols accordantly.

The following image (image 4.8) has been shot from a supporting application that was developed for testing and evaluating purposes. In this image are obvious most of the components characteristics.

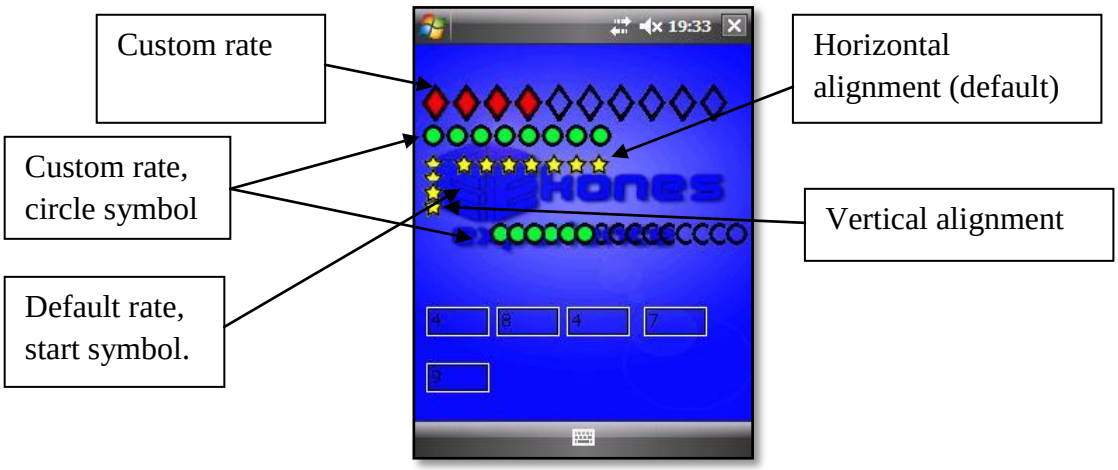


image 4.8 Testing application for istlRate istlRateItem

In previous image are obvious the 3 different user rate images, the alignment and even the distances between the rate elements.

Usage

```
istlRate rt = new istlRate(5);
```

With the usage of the istlRate(5) constructor a 5 star rate item is created, the argument denotes how many stars the user wants the rate item to have initially. After the construction of an istlRate with the usage of additional methods the istlRate component can be modified.

Return type	Method name	Usage
int	getId()	Returns the id value
int	getRate()	Returns the rate
void	setAlignmentVertical()	Sets vertical alignment
void	setCompact(int compact)	Sets desired distances between rate images
void	setId(int id)	Sets id
void	setIstlItemSize(int AspectDimension)	Sets rate image size relatively with the existing.
void	setIstlRateSize(com.sun.lwuit.geom.Dimension d)	Sets the rate image size.
void	setRate(int no)	Sets number of rate icons
void	setRateEmptyIcon(com.sun.lwuit.Image empty)	Sets a custom rate image for empty state
void	setRateFillIcon(com.sun.lwuit.Image fill)	Sets a custom rate image for full state.
void	setRateStyle(com.sun.lwuit.plaf.Style ratestyle)	Set a style for the istlRate component.

Calendar Component Architecture

The calendar component was created with the intention to offer the "time" representation of the package to the user, but also to support other miscellaneous functionalities.

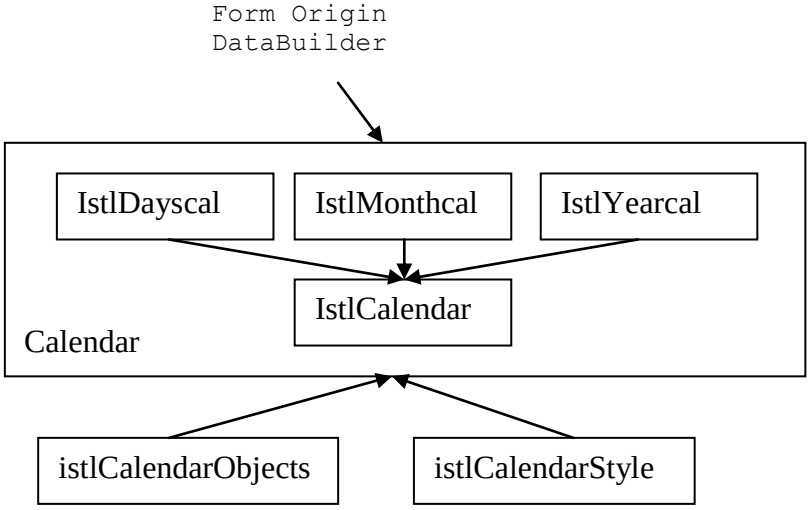


Image 4.9 Calendar component architecture.

The calendar component (image 4.10) consists of 3 sub-components (istlDayscal, istlMonthcal, istlYearcal) and a main container named (istlCalendar which extends the container Form) (image 4.9). The istlCalendar component has a constructor with 2 arguments, the first argument is a Form type and this cause since the istlCalandar component is a Form type, in order to link with the rest application needs an argument of Form type that denotes the source Form which triggers the Calendar component, the second argument is a Databuilder class argument that holds the application data which have derived from the e-kone server with package information.

Εικόνα 4.10 Calendar

The sub-components istlDayscal, istlMonthcal, istlYearcal have specific functionality, therefore the istlYearcal component (image 4.11) extends the LWUIT container and its job is to draw the current year with two buttons for previous / next year shift, similar structure has the istlMonthcal component (image 4.12) only that the next / previous shift is for months. The most complex component of the three is the istlDayscal component (image 4.13), this component extends the LWUIT Container like the previous and has a constructor with three arguments (current month, current year, Databuilder) but at the same time it consists and creates two separate istlContainer components ,one container named cont_day_header is in charge of draws the header with the days names, while the second named cont_dayscont is in charge for drawing the days depending the month and the year, in extend it assigns on every day an ActionListener. Finally it marks with different color the package days.

From the previous it is obvious that the istlCalendar component is hardcoded for the needs of the application.

istlCalendarObjects keeps data for user notes but also do some basic functions with them. The istlCalendarStyle is a theme class that is responsible for the graphical elements and appearance of the component.

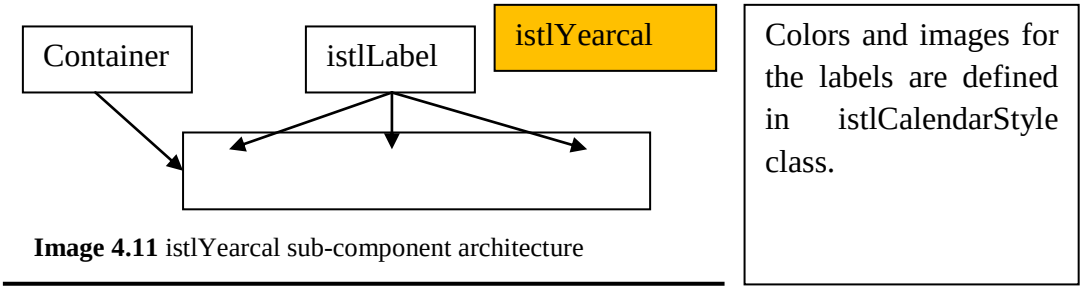


Image 4.11 istlYearcal sub-component architecture

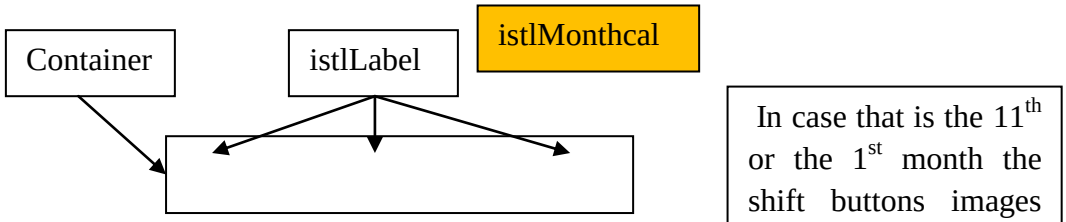


Image 4.12 istlMonthcal sub-component architecture

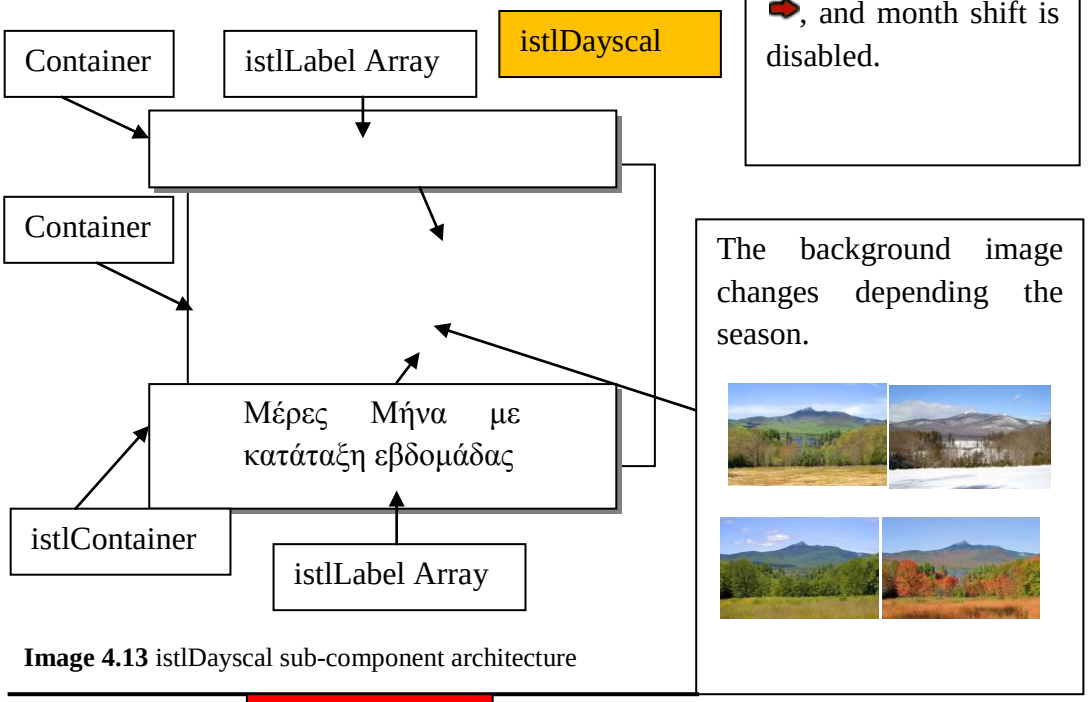


Image 4.13 istlDayscal sub-component architecture

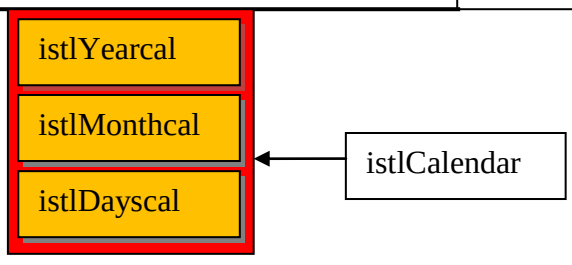


Image 4.14 The general architecture with the 3 sub-components combined.

istlMap Component Architecture

The istlMap component offers spatial representation of the package (image 4.15). The term “spatial” representation means all the information like the Business Partners, other users of the package, and personal information. All these information in order to have the spatial meaning are appearing on a map, which derived from the Static Google Maps service. Also the coordinates of user position are displayed, but not graphically represented.

The istlMap component extends the form component, and has two constructors, one with arguments a component of form type and a Vector data type that holds the business partner data, while the other constructor has in addition a UserActivityData object that has user information. The reason for the 2 different constructors is a J9 limitation which will be described at the end.

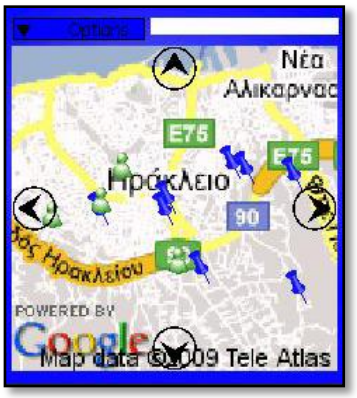


Image 4.15 istlMap

The istlMap component consists of two istlContainers sub-components (image 4.16), one named panel, is on the upper of the screen and includes a two level istlMenuItem as well an istlLabel component in which appears user’s position (image 4.17). The other named map is used as the container on which are placed the map and the marks (user markers, ekones users’ markers and e-kones markers) (image 4.18) and will analyzed in a next chapter.

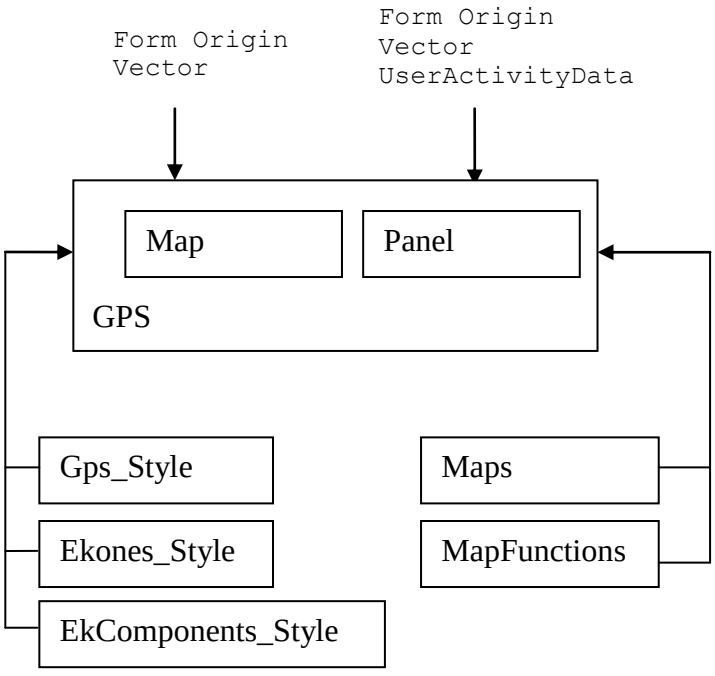


Image 4.16 istlMap Architecture

In total the istlMap component is a complex component due to the interaction abilities that offers, like marker insertion, deletion, move, information projection for the business partners, and tracking the users movement that are registered in the same package. Despite these functionalities this component's complexity is raised more by other algorithms that are executed in order to support the previous, algorithms like the map functions algorithm, and 2 threads that are running for the periodically update of the user position and the other users position on the map are raising the complexity.

The previous make obvious that the istlMap component is hardcoded without generic character.

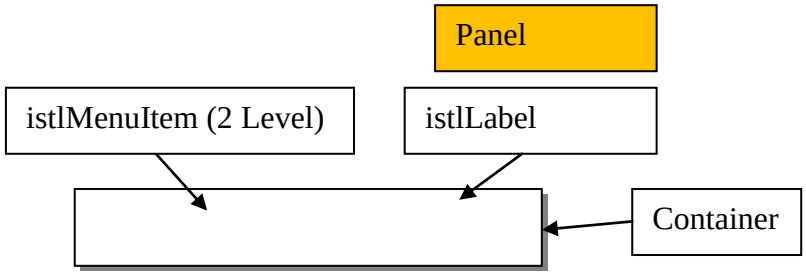


image 4.17 panel subcomponent architecture.

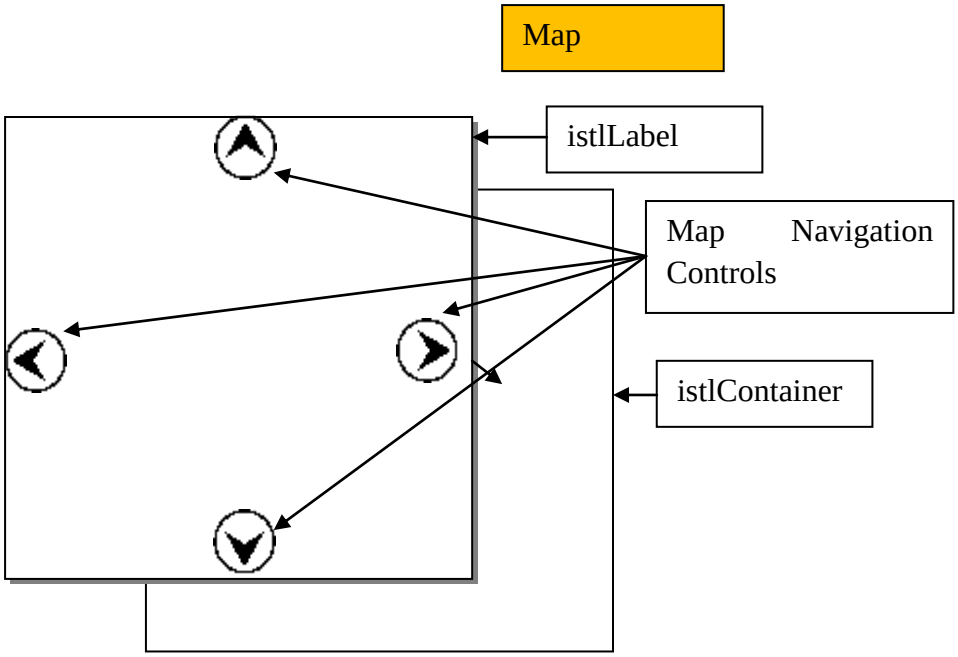


image 4.18 map sub-component arhitecture

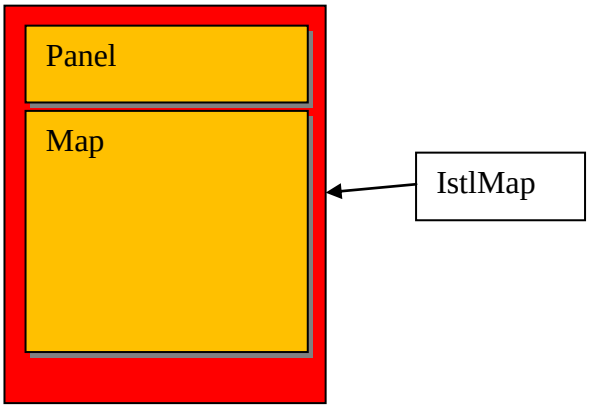


image 4.19 istlMap general architecture with sub-components combined

Interaction with MAP container of istlMap Component

The map component is an istlContainer type, as previously described the istlContainer component can identify "pen" events and with appropriate methods returns the screen coordinates where the event occurred.

The only event that the map container handles is press and drag events on its area. A press event in the place that occurred creates an istlPopUp component offering options to the user, while the drag event causes a user marker to move in an other position on screen.

Markers on Map

The application provides 3 types of markers to the user (user markers, e-kones markers and user position markers). Those markers aren't produced in a dynamic way, rather than they created when the Map component initializes. The number of e-kones markers that are representing a business partner location are derived from the total number of business partner in the XML that the application receives from the e-kones server, after an ActionListener is assigned to each of the marker for each business partner, finally each marker is placed on the map depending the geographic position that each business partner has registered to e-kones service (and send through the XML to the application). The sum of users positions markers that are registered in the same package is computed again from the appropriate XML in a similar way as the business partners markers. The sum of user markers cannot be defined from the beginning, neither can be infinite (for performance reasons), there should be a max value for the total number of user markers that the user can place on map therefore the application assumes that the total number for user markers should be equal to the sum of business partners. In user markers an ActionListener is assigned that generates an istlPopUp component with appropriate selections. User markers are placed out of screen, not visible and disabled. In fact when the user inserts a marker, the application enables that marker and positions it in the map position that a press pen event occurs, in similar way when a user deletes a marker the application disables it and move it off screen.

Each marker object is an istlLabel component which generates its own istPopUp when clicked and complies to its own limitation (example the user can't delete an e-kones marker).

Threads

The istlMap component starts executing two threads after its initialization. One thread is responsible for the periodically update of the application with the users position that are registered in the same package. Every 10000 millisecond the thread updates from the e-kones server with the new users position and place the users positions markers in the appropriate position on map. The second thread runs every 1000 millisecond and updates the component with the new user geographical position.

Please note that the service for the users positions hasn't been created therefore the positions that are appearing on the map in this version are derived from a server XML which is stored in a file and not produced dynamically (realtime) from the service. The creation of a service like that in the frame of this project was out of the scope. Despite that a possible architecture for such a service is shown in the following diagram (image 4.20).

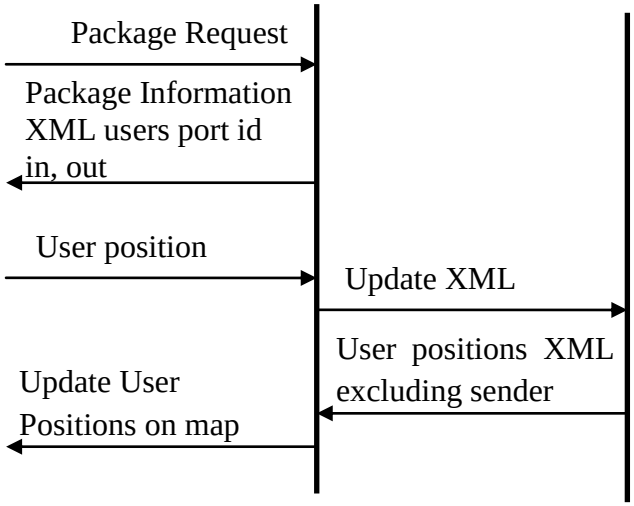


Image 4.20 A possible service XML based architecture that could support the exchange in real time position massages between clients and service.

In this theoretical data exchange model between server and user the creation of a third thread is required in order the user to send his location to the server. The server after the desired package selection from the user, is sending the package information (like in this version) and the two communication ports on which the application will send and receive user position information. During the initialization of the istlMap components the two threads responsible for user position begin to execute, it is logical that will required different execution times for sending and receiving as well and a communication protocol for sending service messages which can configure the execution times in cases like server buffer overflow in order to avoid data loss.

From the server side, when the server receives position data in specific times will refresh the file with the positions, the refresh could be either in specific times

either under a specific case (example, every client has send position update for X times). The server input must have a quite large buffer in order to catch up and make I/O operations with minimum error possibility. Also a message protocol should be established in order to notify the users for occasions of high traffic in order to reduce the sending rate.

An XML sample that it is used from the application for receiving users positions without taking in account a traffic configuration protocol is the following.

```
<users>
    <user id='1'>
        <username>Kotsalomistos</username>
        <last_name>Kotsalis</last_name>
        <first_name>Dimitris</first_name>
    </user>
    <user id='2'>
        <username>Gallactica</username>
        <last_name>Milolidakis</last_name>
        <first_name>Giannis</first_name>
    </user>
    <user id='3'>
        <username>Larisaios</username>
        <last_name>Vellis</last_name>
        <first_name>Giorgos</first_name>
    </user>
    <user id='4'>
        <username>Episimonas</username>
        <last_name>Plemenos</last_name>
        <first_name>Argiris</first_name>
    </user>
</users>
```

This XML can be modified in order to include traffic configuration messages.

```
<exchange>
<code>
     <transmition_rate>1</transmition_rate>
</code>
<users>
    <user id='1'>
        <username>Kotsalomistos</username>
        <last_name>Kotsalis</last_name>
        <first_name>Dimitris</first_name>
    </user>
    <user id='2'>
        <username>Gallactica</username>
        <last_name>Milolidakis</last_name>
        <first_name>Giannis</first_name>
    </user>
    <user id='3'>
        <username>Larisaios</username>
        <last_name>Vellis</last_name>
        <first_name>Giorgos</first_name>
    </user>
    <user id='4'>
        <username>Episimonas</username>
        <last_name>Plemenos</last_name>
```

```
        <first_name>Argiris</first_name>
    </user>
</users>
</exchange>
```

The XML above has been modified appropriately in order to support system messages exchange. The XML node `<code></code>` can hypothetically encapsulates system messages, in addition the attribute `<transmition_rate> </transmition_rate>` which is in the node `<code></code>` could hold information related to the transmition rate, a value of 1 like in the example could mean normal transmition rate, while greater than 1 could have a negative rate factor example a value of 5 could be a reduce by 50% on the previous transmition rate, and a value of 0 could mean out of order. The application based on these values could send the user position to the server.

Algorithms for Map calculations

One of the most difficult tasks for this project was the conversion of geographic coordinates to screen coordinates, with the usage of a map from the Google Maps Service and the ability of changing the zoom level, resolution and map type.

The Google Maps service is offering maps based on a request map center, with desired zoom, resolution and type. Given the known screen size, the geographical position of the business partners (derived from the XML), the geographical position of the user derived from the GPS system of the device, the appropriate position of these on the screen must calculated. The container on which the map will be placed has resolution of 232X232 pixels.

The system calculates the center based on the geographical coordinates of the business partners, and requests from the Google Maps service a map with that center. With an experimental way was found the mathematical relation between zoom and resolution.

Resolution	Zoom	Shift Center	Shift Side
128X128	17 (max)	0.000689	0.001378
	8	$0.000689X2^{9}$	$0.001378X2^{9}$
	4	$0.000689X2^{13}$	$0.001378X2^{13}$
136X136	17 (max)	0.000738	0.001466
	12	$0.000738X2^{4}$	$0.001466X2^{4}$
	8	$0.000738X2^{9}$	$0.001466X2^{9}$
	4	$0.000738X2^{13}$	$0.001466X2^{13}$
144X144	17 (max)	0.000769	0.001538
	16	$0.000769X2^{1}$	$0.001538X2^{1}$
	12	$0.000769X2^{4}$	$0.001538X2^{4}$
	8	$0.000769X2^{9}$	$0.001538X2^{9}$
152X152	17 (max)	0.000819	0.001638
	16	$0.000819X2^{1}$	$0.001638X2^{1}$
	12	$0.000819X2^{4}$	$0.001638X2^{4}$
160X160	17 (max)	0.0008595	0.001719
	16	$0.0008595X2^{1}$	$0.001719X2^{1}$

	12	$0.0008595X2^4$	$0.001719X2^4$
168X168	17 (max)	0.000903	0.001806
	16	$0.000903X2^1$	$0.001806X2^1$
	12	$0.000903X2^4$	$0.001806X2^4$
176X176	17 (max)	0.000942	0.001884
	16	$0.000942X2^1$	$0.001884X2^1$
	12	$0.000942X2^4$	$0.001844X2^4$
184X184	17 (max)	0.000988	0.001976
	12	$0.000988X2^4$	$0.001976X2^4$
	8	$0.000988X2^9$	$0.001976X2^9$
	4	$0.000988X2^{13}$	$0.001976X2^{13}$
192X192	17 (max)	0.001031	0.002062
	8	$0.001031X2^9$	$0.002062X2^9$
	4	$0.001031X2^{13}$	$0.002062X2^{13}$
232X232	17 (max)	0.00124	0.00248
	12	$0.00124X2^4$	$0.00248X2^4$

In the table above the experimental results data that relate the zoom level with resolution are presented. The 1st table column denotes the map resolution, the 2nd the zoom level, while the third column the shift factor in degrees that is required to retrieve a map with a center the side of the current, the 4th column denotes the required shift in degrees in order to take a whole new map which is the next of the current (as it's logical this is the double value of the 3rd column). The above was experimental results and may contain a big possibility of error.

The fact worth mentioning is that there is a mathematical relation between a specific resolution and zoom level. Unfortunately as it is obvious there is no relation between different resolutions and zoom levels. With a result the calculation of a new marker position to be problematic in cases where the user switches from one resolution to another.

Therefore can be algorithm that calculates the position on the map given a specific resolution.

```
double pxf;
        double long_dif,lat_dif;
        int zoom_dif = Math.abs(16-zoom);
        int zoommf=2;

        for(int x=0;x<zoom_dif;x++)
        {
           zoommf=zoommf*2;
        }
        cfs = 0.00248*zoommf;
        cfc = 0.00124*zoommf;
        pxf=cfs/232;
        long_dif=cen_long+cfc;
        lat_dif=cen_lat+cfc;

        mark_long =  (int) ((long_dif-longt)/ pxf);
        mark_lat = (int) ((lat_dif-lat) / pxf);
```

Except the limit between variable resolution and zoom level a number of other problems occurred mainly cause of LWUIT and J9 incompatibility. Therefore the resolution for map requests locked to a 144X144 resolution, with 12 zoom level, and mobile maptype, which are safe configurations with the fewer problems.

Above the map there are for arrows in each side which enable the map scroll with the usage of a similar algorithm. Unfortunately a problem with calculation of the new marker position arises which make the use of these arrows a bit buggy.

```
    public double
MapRightScroll(double clat)
    {

       int zoom_dif =
Math.abs(17-zoom);
       int zoommf=1;

       for(int
x=0;x<zoom_dif;x++)
       {
           zoommf=zoommf*2;
       }
       cfc = 0.00124*zoommf;
       cen_lat= clat+(cfc/2);
       return cen_lat;
    }
```

```
     public double
MapLeftScroll(double clat)
    {

       int zoom_dif =
Math.abs(17-zoom);
       int zoommf=1;

       for(int
x=0;x<zoom_dif;x++)
       {
           zoommf=zoommf*2;
       }
       cfc = 0.00124*zoommf;
       cen_lat=clat-(cfc/2);
       return cen_lat;
    }
```

```
public double
MapUpScroll(double clong)
    {
       int zoom_dif =
Math.abs(17-zoom);
       int zoommf=1;

       for(int
x=0;x<zoom_dif;x++)
       {
           zoommf=zoommf*2;
       }
       cfc = 0.00124*zoommf;
       cen_long=clong+(cfc/2);
       return cen_long;
    }
```

```
public double
MapDownScroll(double clong)
    {

        int zoom_dif =
Math.abs(17-zoom);
        int zoommf=1;

        for(int
x=0;x<zoom_dif;x++)
        {
           zoommf=zoommf*2;
        }
        cfc = 0.00124*zoommf;
        cen_long=clong-(cfc/2);
        return cen_long;
    }
```

An other algorithm is the algorithm which calculates the distance between business partners. The algorithm is based on Haversine formula.

R = earth's radius (mean radius = 6,371km)
$\Delta lat = lat_2 - lat_1$
$\Delta long = long_2 - long_1$
$a = \sin^2(\Delta lat/2) + \cos(lat_1).\cos(lat_2).\sin^2(\Delta long/2)$
$c = 2.\text{atan2}(\sqrt{a}, \sqrt{(1-a)})$
$d = R.c$

A method was created that executes the required operations for the algorithm. Because the Math class of the Midp doesn't support all the required mathematical

functions a new class created named istlMath which offers the appropriate functions atan, atan2, toRad and Round

```
public double MarkerDistanceCalculation(double slat,double
slong,double dlat,double dlong)
    {
            double R = 6371; // km
            double dLat = istlMath.toRad(slat-dlat);
            double dLon = istlMath.toRad(slong-dlong);
            double a = Math.sin(dLat/2) * Math.sin(dLat/2)
+Math.cos(istlMath.toRad(dlat)) *      Math.cos(istlMath.toRad(slat))
*Math.sin(dLon/2) * Math.sin(dLon/2);
            double c = 2 * istlMath.atan2(Math.sqrt(a), Math.sqrt(1-
a));
            double d = R * c;

            return istlMath.Round(d);
    }
```

Google Maps Communication Component

In order to support the Google Maps service a class was created named Map. That class uses an “empty” constructor.

```
public Maps()
 {

   }
```

With the usage of methods a connection is established with the Static Google Maps service and a map is retrieved. The class has method that connects with the service using the http protocol. In order to be a successful connection and map retrieval some more information are required, like the service url, resolution, zoom level, map type, those functionalities are handled by extra methods. An example for connection with the Google Maps service is the following.

```
mp = new Maps();
mp.setMapZoom(cfg.getZoom());
mp.setMapXdimension(cfg.getResolution());
mp.setMapYdimension(cfg.getResolution());
mp.setMapProfile(cfg.getMapType());
mp.setMapLon(rd[0]);
mp.setMapLat(rd[1]);
mp.setMapURL();
```

In the previous code sample an instance of the class Maps is created with the name mp. In this example a class named GpsConfig appeared, from which return values from its methods are used as arguments for the methods of the mp instance, as shown above.

```
cfg = new GpsConfig();
cfg.setMapType("mobile");
cfg.setResolution(144);
cfg.setZoom(12);
```

The GpsConfig class keeps user values for the desired map (In the final prototype version due to the reason motioned before that class hasn't much usage).

The map properties have been sent, now the final step is to actually connect with the service and retrieve the map.

```
Image image;
        try {
            image = Image.createImage(mp.mapConnect(), 0,
mp.mapConnect().length);
        } catch (Exception ex) {
            istlmapholder.setText(ex.getMessage());
        }
```

The connection has been establiehd and a map has been retrieved, the map is placed inside the container as explained before and shown to the user.

Class Maps methods

Return type	Method name	usage
com.sun.lwuit.Image	getMapImage()	Return the image of the map
java.lang.String	getMapUrl()	Return google maps url
byte[]	mapConnect()	Connects with the service and returns the map.
void	setMapDimensions(com.sun.lwuit.geom.Rectangled)	Sets map resolution
void	setMapFormat(java.lang.String format)	Sets the map format (.png8, .png24).
void	setMapKey(java.lang.String key)	Sets users unique key, given by the google maps service
void	setMapLat(double lat)	Sets the desires latitude to use a center for the map
void	setMapLon(double lon)	Sets the desires longtitude to use a center for the map
void	setMapProfile(java.lang.String profile)	Sets the desired map type
void	setMapURL()	
void	setMapURL(java.lang.String url)	
void	setMapXdimension(int x)	Sets the map resolution (x dimension)
void	setMapYdimension(int y)	Sets the map resolution (ydimension)
void	setMapZoom(int zoom)	Sets the desired zoom level

GPS Component

The application uses the NMEA protocol in order to receive data form GPS chipset on the device (SirfStarIII). The communication with the GPS chipset is done through an appropriate class that has been created, named GPS. The application uses a data class named GpsPosData in order to hold the GPS data, and a class named GPS which is responsible for reading massages from the port.

At this point must be noted that a full GPS application should use every GPS message key that the GPS chip sends, from simplicity reasons this application utilizes only messages with the $GPGGA message key which holds position data.

Message Key	Message Type
GGA	Time, position and fix type data.
GLL	Latitude, longitude, UTC time of position fix and status.
GSA	GPS receiver operating mode, satellites used in the position solution, and DOP values.
GSV	The number of GPS satellites in view satellite ID numbers, elevation, azimuth, and SNR values.
MSS	Signal-to-noise ratio, signal strength, frequency, and bit rate from a radio-beacon receiver.
RMC	Time, date, position, course and speed data.
VTG	Course and speed information relative to the ground.
ZDA	PPS timing message (synchronized to PPS).
150	OK to send message.

The architecture that is responsible for reading GPS messages from the device is shown below (image 4.21).

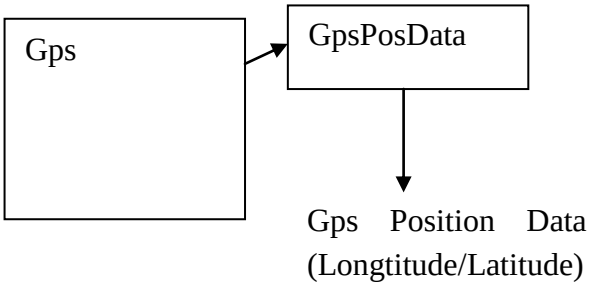


image 4.21 Component architecture that communicates with the GPS

An alternative implementation that could support more messages could be like the following (image 4.22).

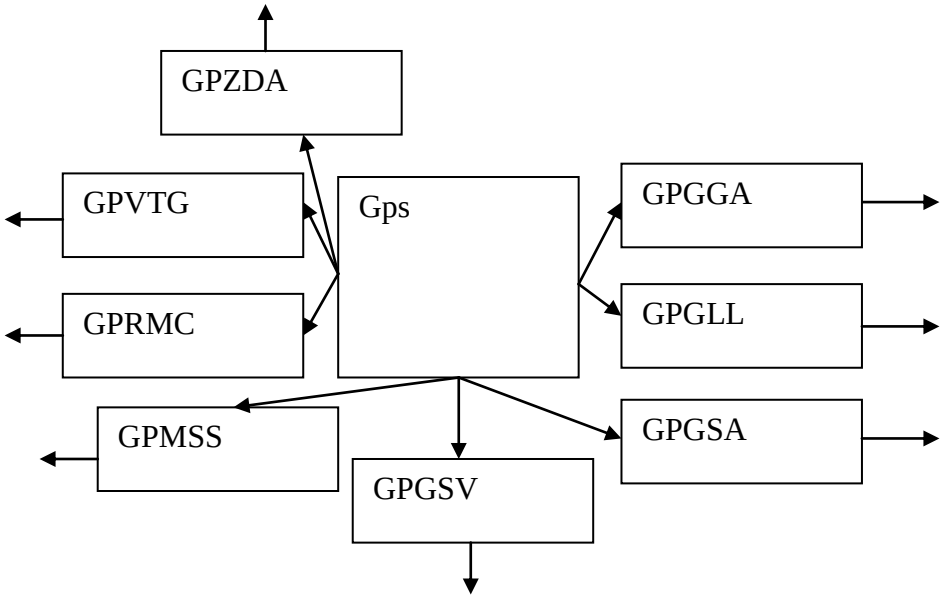


image 4.22 An alternative GPS component architecture that could utilize all of the GPS

The architecture shown above utilizes in full the GPS chipset, along with an appropriate classes that could carry the NMEA message data.

In order the GPS class to function first needs to initializes.

```
GPS  gp = new GPS();
```

When an instance of the GPS class is created, automatically connects with the com4 port trying to read messages with the $GPGGA message id, which are having position information. After a GpsPosData object is constructed and with appropriate methods it is filled with data. The application can retrieve the data, with the following way.

```
gp.RetreiveGpsData();
 GPSPosData gpd =gp.getGPSPosData();
```

Now the application with the appropriate methods that the GpsPosData class provides can handle the data.

Return type	Method name	usage
java.lang.String	getGps()	Return a full unprocessed message line
java.lang.String	getKey()	Return the message key.
java.lang.String	getLat()	Returns the latitude
java.lang.String	getLong()	Returns the longitude
int	getNoOfSatellates()	Returns numbers of satellites covering the area
double	getTime()	Returns time
boolean	isValid()	
void	setGps(java.lang.String gps)	
void	setKey(java.lang.String	

	key)	
void	setLat(double Lat, java.lang.String Dir2)	
void	setLong(double Long, java.lang.String Dir1)	
void	setNoOfSatellates(int noOfSatellates)	
void	setTime(double time)	
void	setValid(boolean valid)	

In order for the application to have continues stream with position data, a thread has been created that every 1000 millisecond returns position data.

In e-kones experience application the abilities of this component aren't well show, therefore a small application that shows in more complete the abilities of this component has been created (will examined in the appendix).

Application Data Management

The Application due to its nature (client application for E-kones Application) demands connectivity with a server through the internet, while it is also required access on the device file system.

Device File System Access

The application demands the data transfer from form to form, in order to keep persistent the user data during the runtime but also after the termination. As we've already mention the IBM J9 hasn't any optional package implementation, in order a J2ME Midp application to have access on the file System of the device the support of JSR-75 is required (File Connection – PDA Optional Package).

For applications that demand data storage without JSR-75 support the Midp 2.0 offers the so called "Record Store" (RMS – Record Management System). The RMS is in a way an internal database that keeps inside the implementation information for data storage, it has been created and used mainly for storing simple data like scores for games. This specific way for storing data could serve as a way to store user preferences up to a degree, but it couldn't serve the needs for "temporary files" which this application demands.

The lack of a storing way lead in the creation of "heavy" constructors which are moving data from a class to an other in order to keep the data persistent. This has as a result a great performance impact. Even though his trick couldn't solve the problem, therefore the application prototype keeps persistent data only between a number of components.

Data Access From E-kones Service

At the communication part between the server and the application the Midp offers support for the http protocol. The application uses XML files to communicate with the server. In order to make the XML access possible an XML parser was

created (a small test program that full utilizes and show the abilities of the XML parser will be described in the Appendix)

The Application uses the following categories of data.

- Packets that the user has registered.
- Data specific related to a package.
- Data for users positions and information.
- Personal User Information.
- Configuration Data (Aren't fully implemented).

The architecture for the data is presented in the next diagram (image 4.23).

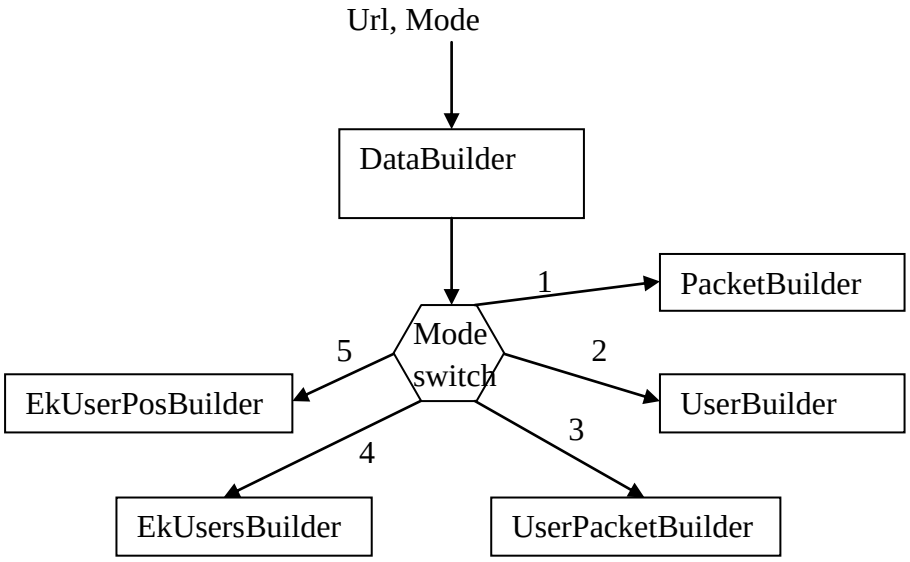


image 4.23 Architecture for the application data.

From the diagram above it is obvious that there is a class with a constructor named DataBuilder which has as argument the url address of the xml file, as well a mode of operation. Depending the selected operation mode a number of methods are executed.

The DataBuilder Constructor.

```
public DataBuilder(String url,int mode)
{
    this.url=url;
    if(mode==1)
    {
       PacketBuilder();
    }
    else if(mode==2)
    {
       UserBuilder();
    }
    else if(mode==3)
    {
       UsersPacketBuilder();
    }
    else if(mode==4)
```

```
    {
       EkUsersBuilder();
    }
    else if(mode==5)
    {
       EkUserPosBuilder();
    }
}
```

As it is shown the constructor provides no insurance for a mode selection out of the predefined as well in cases where the user gives a wrong url.

Mode 1, This operation mode enables methods for building classes with information related to the selected package. The execution code is shown below.

```
public void PacketBuilder()
{
   Days = new Vector();
   xmcon = new XMLConnect(url);
   CreatePackageDescriptor();
   CreatePackageDays();
   CreatePackageLocations();
   CreatePackageCategories();
   CreatePackageActivities();
   CreatePackageBPartners();
   CreatePackageParameters();
}
```

The operation mode is shown in the next diagram (image 4.24).

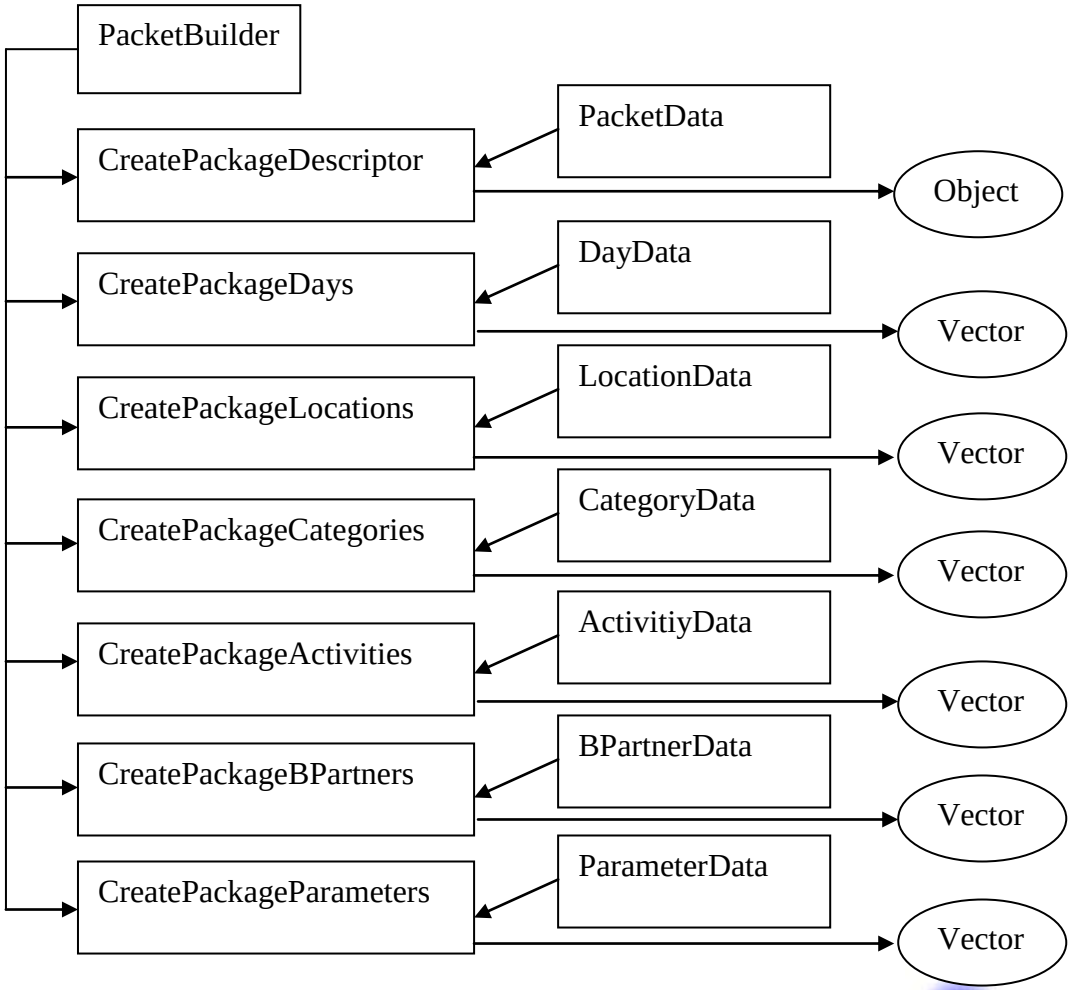


image 4.24 Architecture for building classes for an e-kones package

Each method that the DataBuilder method executes reads the XML and fills a data class with the appropriate data. Each method when finishes data reading produces a data type which contains the data. All of the methods are producing vector data type except the CreatePackageDescriptor method which produces an object data type, derived from the data class. This is logical since there can be only one description for every package, but there can be a lot of activities, business partners etc.

The final data will have the following form (image 4.25).

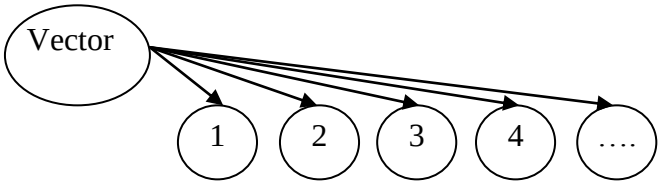


Image 4.25 Data model for a packet

Each element in the Vector data type could be type of activitydata, bpartnerdata, categorydata etc, depending the content data type the Vector has appropriate name, for example the Vector data type that has activitydata as elements and has been created by the CreatePackageActivities, is named Activities.

Mode 2, This operation mode utilizes method for building classes with application user information. The execution code is shown below.

```
public void UserBuilder()
{
   xmcon = new XMLConnect(url);
   CreateUser();
}
```

The operation mode is shown in the next diagram (image 4.26).

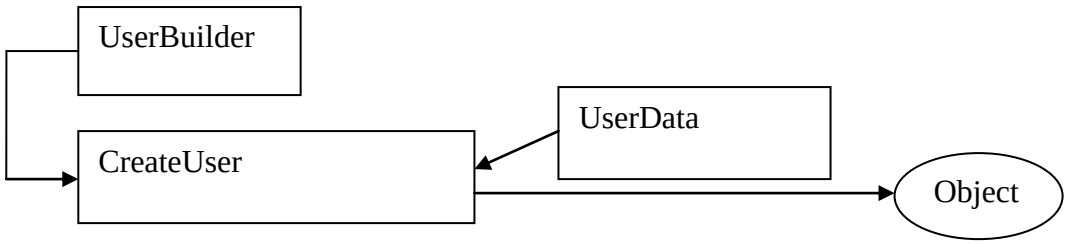


Image 4.26 Architecture for building information classes for e-kones experience user.

The UserBuilder method has only one method which reads the desired XML and puts the data in the specific class. When finishes reading it produces a UserData object which holds the user information.

Mode 3, This operation mode utilizes method for building classes with package information in which the user has registered. The execution code is shown below.

```
public void UsersPacketBuilder()
{
   xmcon = new XMLConnect(url);
   CreateUserPackages();
}
```

The operation mode is shown in the next diagram (image 4.27).

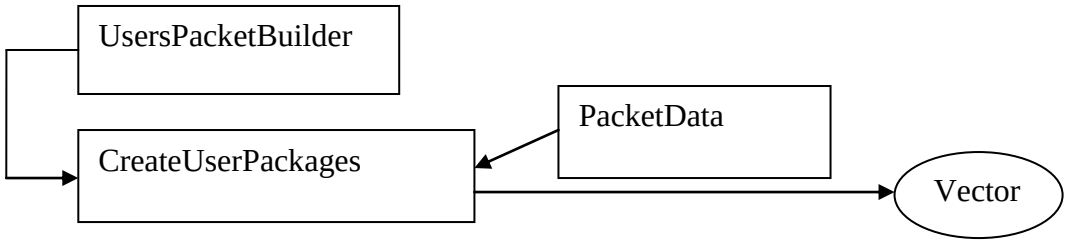


Image 4.27 Architecture for building data classes for packages that the user has subscribed.

The UsersPacketBuilder method has only one method that reads the XML and puts the data in the appropriate data class. At the end of reading it creates a Vector data type that holds package information for every package that the user has subscribed to.

Mode 4, This operation mode utilizes method for building classes with users that are registered to the same package. The execution code is shown below.

```
public void EkUsersBuilder()
{
   xmcon = new XMLConnect(url);
   CreateUsers();
}
```

The operation mode is shown in the next diagram (image 4.28).

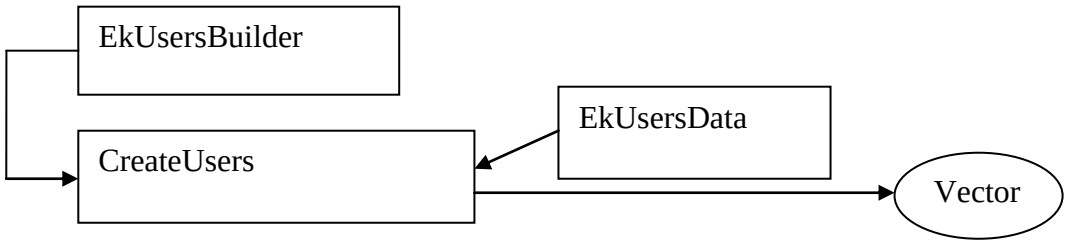


Image 4.28 Architecture for building data classes for the users that are registered at the same e-kones package

The EkUsersBuilder method utilizes a method that is in charge of reading the appropriate XML file and put the data in the appropriate data classes. After the reading the method produces a Vector data type with information for users that are registered to the same package.

Mode 5, This operation mode utilizes method for building classes with users positions. The execution code is shown below.

```
public void EkUserPosBuilder()
{
  xmcon = new XMLConnect(url);
  CreateUserPos();
}
```

The operation mode is shown in the next diagram (image 4.29).

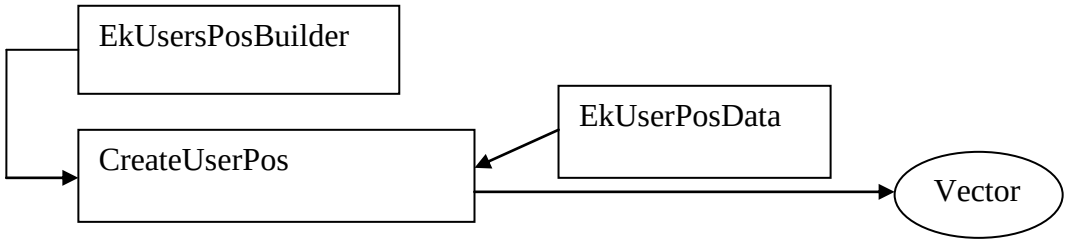


Image 4.29 Architecture for building data classes with users position information.

The EκUsersPosBuilder utilizes a method that reads the XML file and fills with data the appropriate classes. When the reading is finished it produces a Vector data type with EkUserPosData as elements and holds every user position.

During the application execution the DataBuilder object works as follows.

```
DataBuilder upkb;
upkb=new
DataBuilder("http://hwm.armedassault.info/personal/XMLPackageServlet1
.xml",1);
```

A DataBuilder object is created with the name "upkb", and arguments the XML url and the operation mode. The operation mode here is "1" which means that the method will executes the part responsible for building specific Package information data classes. After the creation, the application can access the data with the following way.

```
Vector v = new Vector();
v = upkb.getPackageDays();
```

In the example above the getPackageDays() method returns a Vector type object that will has DayData type as elements, with information for every day of the package. Each element of the vector can be accessed with the following way.

```
for(int k=0;k<v.size();k++)
{
  DayData dt = new DayData();
  dt=v.elementAt(k);
  String str = dt.getDescription();
  System.out.println(str);
}
```

In the example above every data that the Vector (named v) data type holds are read, after each element placed in a DayData class instance named "dt" and a method retrieves the description for every day of the package.

Methods provided by the DataBuilder class

Return type	Method name	usage
`void`	`EkUserPosBuilder()`	Central method calls methods for building users position data classes

void	EkUsersBuilder()	Central method calls methods for building users information data classes
java.util.Vector	getEkonesUsersData()	Return user Data
EkUserPosData	getEkonesUsersPositionData()	Returns users position data
java.util.Vector	getPackageActivities()	Returns package activities data
java.util.Vector	getPackageCategories()	Returns package categories data
java.util.Vector	getPackageDays()	Επιστρέφει τι ημέρες του πακέτου.
java.util.Vector	getPackageLocations()	Return package location data
java.util.Vector	getPackageParameters()	Return package parameter data
java.util.Vector	getPackagePartners()	Returns package business partner data
PacketData	getPacketDescriptor()	Returns package general description
UserData	getUserData()	Return user info data
java.util.Vector	getUserPackages()	Returns packages that the user is registered to.
void	PacketBuilder()	Central method calls methods for building packet data classes
void	UserBuilder())	Central method calls methods for building user information data classes
void	UsersPacketBuilder()	Central method calls methods for building users information that are registered in the same package data classes

Data Classes.

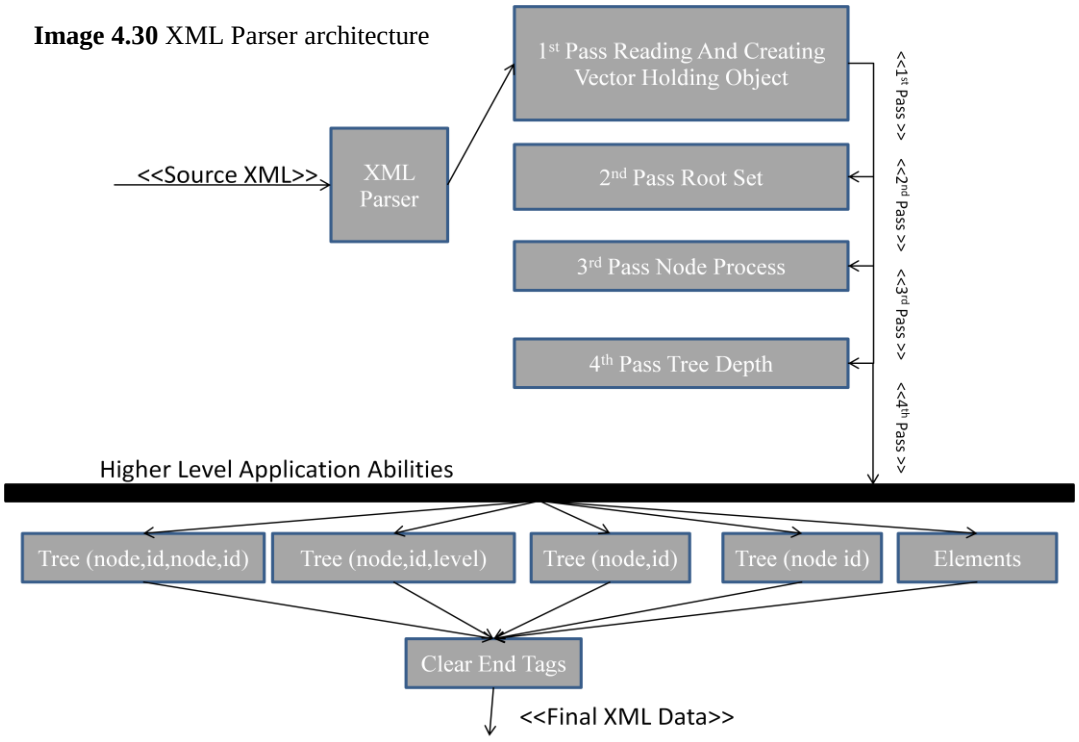
Data class name	Function
ActivityData	Holds activity data
BPartnerData	Holds Business Partner Data
CategoryData	Holds data for package categories
ConfigMapData	Hold map data (not used)
DayData	Holds data for package days
EkUserPosData	Holds data for users position
EkUsersData	Holds data for users registered to the same package
LocationData	Holds data for package locations
PacketData	Holds data for packets that the user is registered to.
ParameterData	Holds package parameter data
UserData	Holds user info data

Each data Class offers different methods, based on its data. All methods in these data classes are set/get types and isn't exist any calculation method.

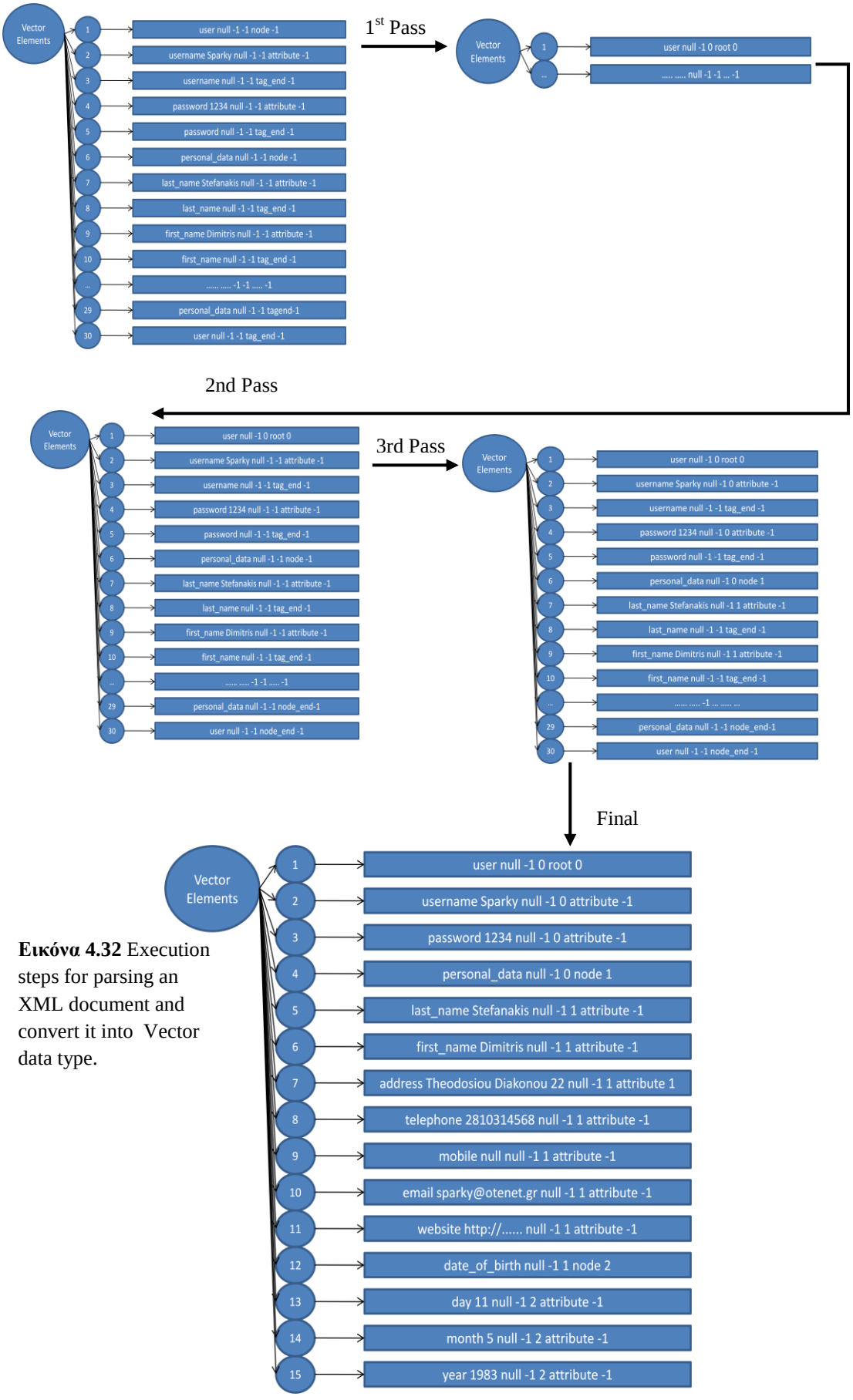
XML Parser

A basic component of this application is the XML Parse. The created parser can read XML files of any kind with a good tolerance on errors. The XML file is read serially, from start to the end of the file. While parsing it tries to create a tree structure for easy data access from the higher levels using a vector data type. Due to the way that this parser is reading data it's not suitable for very large files. For the development of the XML parser it was created a small test application (tool) which will be described at the Appendix. The architecture of the parser is presented in the following diagram as well with an example (image 4.30).

Image 4.30 XML Parser architecture




```
<user>
	<username>Sparky</username>
	<password>1234</password>
	<personal_data>
		<last_name>Stefanakis</last_name>
		<first_name>Dimitris</first_name>
		<address>Theodosiou Diakonou 22</address>
		<telephone>2810314568</telephone>
		<mobile>null</mobile>
		<email>sparky@otenet.gr</email>
		<website>http://ofp.gamepark.cz/_hosted/mmstudios</website>
		<date_of_birth>
			<day>11</day>
			<month>5</month>
			<year>1983</year>
		</date_of_birth>
	</personal_data>
</user>
```



Εικόνα 4.32 Execution steps for parsing an XML document and convert it into Vector data type.

Example of usage by the application.

```
xmcon = new XMLConnect(url);
```

An XMLConnect object is created with a URL of the XML document as argument.

The following code example is actually from the application and used to create ActivityData classes and fill a vector with them. Here will be presented the methods of the higher level that offers easy ways to handle the data.

```
private void CreatePackageActivities()
   {
……..
      xmlp = xmcon.getXMLTree("activities","null",0);
      int[] activities_id;
      int activities_count=0;

      while (x<xmlp.size())
      {
         xm = (XMLData) xmlp.elementAt(x);
         if(xm.getTag().compareTo("activity")==0)
         {
            activities_count++;
         }
         x++;
      }
……..
…….
      for(i=0;i<activities_count;i++)
      {
         xmlp =
xmcon.getXMLTree("activity",String.valueOf(activities_id[i]));
         ad = new ActivityData();
         x=0;
         while (x<xmlp.size())
         {
            xm = (XMLData) xmlp.elementAt(x);
            ad.setActivityId(activities_id[i]);
            if(xm.getTag().compareTo("description")==0)
            {
               ad.setActivityDescription(xm.getContent());
            }
            if(xm.getTag().compareTo("departure")==0)
            {
               k=0;
               if(xm.isNode())
               {
                  xmlp2 = xmcon.getXMLTree(xm.getNodeId());
                  while (k<xmlp2.size())
                  {
                     xm2 = (XMLData) xmlp2.elementAt(k);

ad.setActivityDeparture(Integer.parseInt(xm2.getContent()));
                     k++;
                  }
               }
            }
…….
……..
……
         Activities.addElement(ad);
      }
   }
```

This method returns a vector from the XML with the name “Activities” as root.

The xm.getTAg() returns the tag name.

The method xm.isNode() checks if the element is Node type.

The method xm.getContent() returns the content of an attribute.

Method similar with the first helps to return a subtree from the total vector, but not with a string argument but with id. The parser during the parsing gives NodeIds to nodes and RootIds to nodes and attributes. The basic difference is that a node, will have a nodeId and probably a RootId, an attribute can have only RootId since the NodeId for the attribute is always -1 That difference is the basic logic for creating the tree structure of the data in this application.

The obvious thing from the previous example is that in this level with the usage of simple methods we can take the data needed.

Return type	Method name	usage
java.util.Vector	getTree(int root)	Returns a tree structure based on id
java.util.Vector	getTree(java.lang.String tag_name , java.lang.String id)	Returns a tree structure based on a string value for the root and the id
java.util.Vector	getTree(java.lang.String tag_name , java.lang.String id, int level)	Returns a tree structure based on a string value for the root ,the id and the desired level.
java.util.Vector	getTree(java.lang.String tag_name , java.lang.String id, java.lang.String where, java.lang.String where_id)	Returns a tree structure based on a string value for the root the id, the string of a father that contains that tree and the id
java.lang.String	getId()	Returns the id
int	getNodeId()	Returns the node id
int	getRootId()	Returns the root id
java.lang.String	getTag()	Returns the tag title
java.lang.String	getType()	Returns type (node, attribute, root).
boolean	isAttribute()	Checks if this is attribute
boolean	isContentNull()	Checks if content is null
boolean	isNode()	Check if it's a node

CHAPTER 5

Use Case Scenario for E-kones Experience

Here a use case scenario will be presented, starting of how the user can connect to the e-kones service, select a package and retrieve a package in which the user is registered to, as well the functionality that the application offers in order the user to interact with the package.

Connecting to e-kones Service

Image 5.1 login form.

During start up the user is prompted to give a username and password (image 5.1). The application checks if the user exists in the service (has registered at least in one package) and sends to the application the XML file with the packages that the user has registered to as well and his personal information.

Image 5.2 Main form.

In case of a success login (if the user exists in e-kones database) the main form appears (image 5.2). At this point the application knows about who the user is, therefore a welcome message appears on top of the screen along with the current time. The noticable thing is that there are only 3 options available (UserInfo, Currency Calculator, Select Package) and the application prompts the user to select a package.

Package Selection

In order to select a package the user must click on the package icon  from the main menu.

The next interface (Package selection form) demonstrates all the packages in which the user has registered to (image 5.3). Information that appears for each package are: the package icon, title, and date of start and end. Each package has an istlMenuItem that offers two options (Package Information and Select) (image 5.4).

Image 5.3 Package selection form.

The Package Information option provides information about the package, as shown on the right sreen (Package Selection Form)(image 5.5). The provided information are from top to bottom: package title, description, duration in days, and more specific start and end date. Close button terminates that form and returns to the Package Selection form (image 5.3).

Image 5.4 Package selection form. stlMenuItem selections

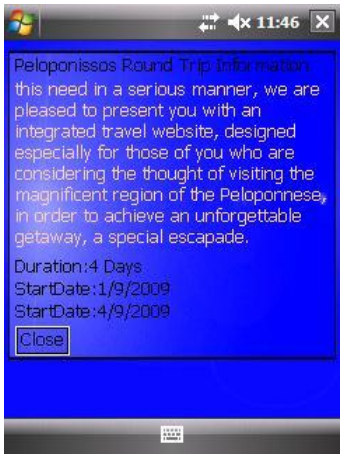


Image5.5 Package Information form

The Select option from the istlMenuItem selects the package and terminates the package selection form.

Image 5.6 Main form

At this point all the buttons in the main form are enabled, and the application informs the user about the active package (image 5.6). From all the buttons the only one with "decoration" character is the Currency Calculator which hasn't been implemented. The other will be explained starting from the simplest.

Personal User Information

Image 5.7 User Information Form

If the user selects the User Info Icon  the User Information form appears (image 5.7). This form provides personal user information like name, last name, address, telephone etc. Close button terminates this form and returns to the main selection form.

Other Users information that are registered to the same package

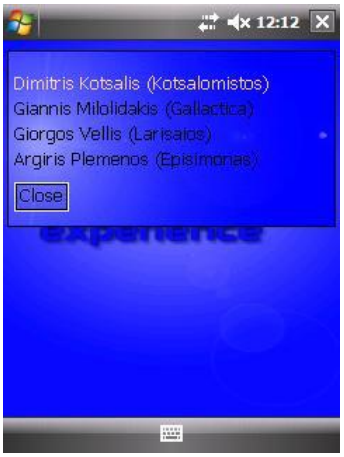


Image 5.8 Users information form.

The User Info icon  initializes the users information form with information about other users who are registered to the same package (image 5.8). E-kones service sends the users XML file immediately after the user has selected a package, that's why this option is disabled until the user selects a package.

Calendar Use

Image 5.9 Calendar

Like previously described the Calendar component offers "Time" representation of the package (image 5.9). The Calendar form initiates by pressing on the Calendar icon  .

When the Calendar form stars by default shows the current year /month. As analyzed before the Calendar component consists of 3 sub-components.

Year sub-component

Image 5.10 Year sub-component

The year sub-component presents the current year and offers two shift buttons. The left /right arrow shifts by one year before/ after from the displayed year (image 5.10).

Month sub-component

Image 5.11 Month sub-component

The month sub-component displays the current month along with two shift buttons. The left/right arrow shifts by one month before/after from the displayed month (image 5.11). In case that the month is December or January the shift arrow becomes red and doesn't not allows any further shift.

Days of Month sub-component

The central part of the Calendar component is the Days sub-component, which is modified based on the current month / year. (image 5.12).

Image 5.12 Days of the month sub-component

On top the initials letters for each day appears starting from Sunday and ending to Saturday. The days are positioned inside the sub-component based on the current days of the month / year. The background picture represents the season, in total there are 4 season backgrounds which are changed dynamically when the season changes. Each day offers interaction with the user, for this use case scenario the 22 of July will be selected.

Image 5.13 Day Form

Image 5.14 Personal Note Form.

Image 5.15 Personal Note Form with example data

Day selection will initiate the Day Form, this form provides information about a package day (if the user is on one of the package days) or a user note (if the user has made one). On the left image the Day form appears empty since this day isn't a package day neither the user has added a personal note.

The Day form consists of the title on top, in this case the generic title E-kones Experience and the current date which is below the title. Left and right of the date are date shift buttons (Haven't implemented). Just below the date there is the central container which is empty offering only the Close and Add buttons (image 5.13).

When the user presses the add button, the New Note Form initiates like shown in image 5.14. In this form the user can add his personal note about this day. The Personal Note Form offers title and Description insertion as well the ability to assign this notation on the map with a marker by using the button (Mark On Map).

After the user has added his personal note, he can submit it to the system by clicking on the Apply button, the Close Button terminates this form and returns to the Day Form. In this example a note is added like the one appearing in the left screen (image 5.15).

The user note is now visible on the Day Panel (image 5.16). Please note the lack of edit, delete buttons which is obvious, isn't implemented mainly due to time limits.

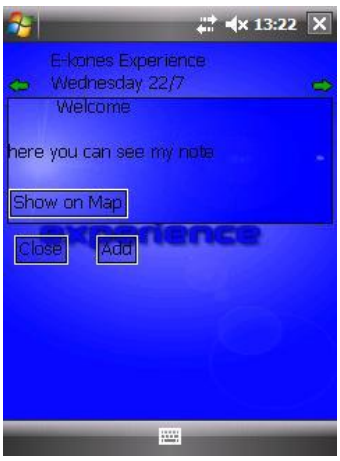


Image 5.16 Updated Day Panel with user Personal Note

The user can add an other note by pressing the Add button (there is no limitation in the amount of user notes) or to click the Close button and returns to the Calendar. In this scenario another note addition isn't required so the Close button is clicked and return to Calendar.

Image 5.17 Updated calendar with the note day marked.

The obvious thing in the Calendar Component is that the day with the user's note has marked with a red color (image 5.17). Generally in this application there are two mainly colors, the red color is for user's notes and the blue for E-kones.

To proceed with this scenario a shift must be made by two months until September where the package is taking place (Note: shifting to the appropriate package month may appear a bit strange. In fact it is not, given the fact that the Calendar starts at the current month and that the user will use the application at the time of the package, the application will start from September. Therefore a shift to the appropriate month will not required. In image 5.18 the following things are noticeable.

Image 5.18 Package days marked with blue color on the calendar.

1. Package Days are marked with blue color (like previously written).
2. The background image has changed providing an image of the month's season.

Clicking on the 1st of September the already known Day Form appears. The difference here is that the Day Form now is not empty. The title has changed from the generic "E-kones Experience" title to the Package title. Another difference is in the main container of the Day form which now has a description of the package. The container now has an extra button named "Activities", clicking that button will imitates the Activities Form with information about the Activities for that Day of Package.

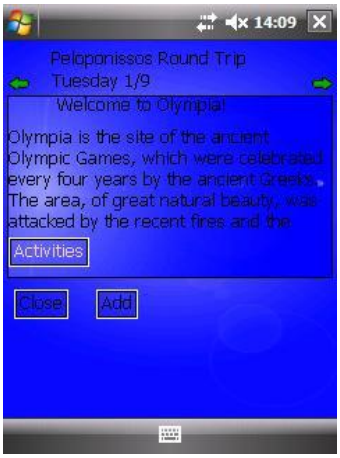


Image 5.19 Day Panel with a Package Day

Εικόνα 5.20 Activities Form.

Image 5.20 shows the Activities Form. On top is the Package title, below the title is the Date along with the number of day which this day is from the package. In main container each activity is encapsulated inside its own container. In this case which the activities are quite few with a result not to fit on a single screen, a scroll bar appears on the left of the screen providing the ability to the user to scroll down to the screen (image 5.21). Each

Image 5.21 Activities Form.

activity container includes the following information.

1. Activity Title.
2. Information about the Activity like price, arrival location, time of arrival etc.
3. The very important istlRateItem that permits to the user to evaluate the Business Partner who provides the Activity.
4. Finally an istlMenuItem that offers a variety of options.

Image 5.22 IstlMenuItem options for this Activity.

In this scenario the last Business Partner will be selected. By pressing the istlMenuItem named Options the following options appearing: Business Partner Information, Location Information, Note, Show On Map, Submit (image 5.22).

Image 5.23 Business Partner Information Form.

The Business Partner Information option initiates the Information Form (image 5.23). From this form the user can see some information related to the specific Business Partner that provides the Activity like telephone number, location, email, address etc. Close button terminates this form and returns to the Activity Form. Location information initiates the Information form for the Location (image 5.24), from where the user can see information about the location where the Activity will take place. Close button terminates this form and returns to the Activity Form.

Image 5.24 Location Information Form.

Image 5.25 Personal Note Form

The Note option opens the Personal Note Form (image 5.25), this specific form is different from the one presented before. The Mark On map button as well the title textfield aren't presented. Since it has no logical meaning for the user to mark on map an already Business Partner's position (it's already marked by the application) either to give title

Image 5.26 Note form with scenario data.

since the note is referring to that Business Partner. For the needs of this scenario the text "quality food will be written" and which will hypothetically express the joy from the user about the food that this Business Partner provided (image 5.26). After by clicking the Apply Button the application updates itself with the note, and finally with the Close button the Note Form terminates and returns to Activity Form.

After the user has put his personal note about this business partner it is also possible to rate this Business Partner. This possibility offers the previously explained istlRateItem. The user can rate the Business Partner by giving from 1 to 5 stars (image 5.27), in this scenario the user gives a 5/5 stars rate. In order to update the rate, from the istlMenuItem the Submit option mu be selected (image 5.28).

Image 5.27 A 5/5 stars rate for this Business Partner.

Image 5.28 Submit rate

The last very important option is the Show On Map option, by selecting this the user can see on map the position of the Bussiness Partner on the map along with his note, rate and other options which will be described later. By clicking on the Show on Map selection the Map component appears (image 5.29). On the map this specific Business Partner is marked with a highlight marker . The user can click on it and see his rate along with his note (image 5.30). From the istlMenu named Options the Close option closes the istlMap component and returns to the Activity Form. The user selects from the istlMenu the Close option.

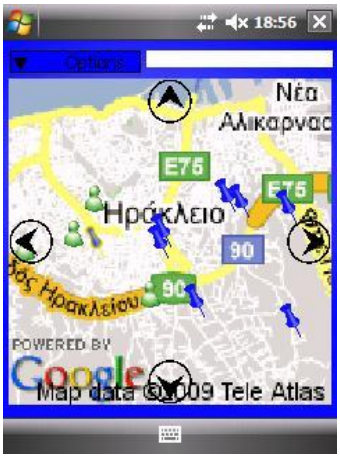


Image 5.29 istlMap with highlighted Business Partner Marker.

Image 5.30 Clicking on the highlight marker provides the user previously input.

Image 5.31 Day Form with Package Description and User Note

The last step before the end of the calendar use case scenario is to add another note in to the Day Form. From the Activities Form at the bottom the Close button terminates the Activity form, and the Day Form appears. There with the way described before by clicking on the add button a new note can be added. In this example the user wrote the note with title "holidays" and the description "so far so good". The obvious in the Days Form is that the user note is put under the Package Description for this day (image 5.31), as previously described the user can add as many notes as he wants but if a day is a package day then every note goes under the package day (as in this example). Close button terminates the Day Form and returns to the Calendar, from there by clicking on the istlMenuItem the user selects the Exit option which is the only available, and returns to the main selection form (image 5.32).

Image 5.32 Calendar, the Exit option terminates the Calendar.

Map Use

Image 5.32 Main Form

From the Main Form the GPS icon activates the map (image 5.32). In istlMap component is noticeable that instead of watching a map with center the user's position, a map with a center an imagine square made by the four most far positions of the Bussiness Partners is displayed, while user's position appears on the top left corner.

The following are noticable (image 5.33).

1. An istlMenuItem with the name Options which offers extra options to the user.
2. The main container with the map image along with the map shift buttons (arrows on the map sides).
3. Different types of markers, during the start the only visible markers are e-kones markers (blue color) and users positions markers (green figures).

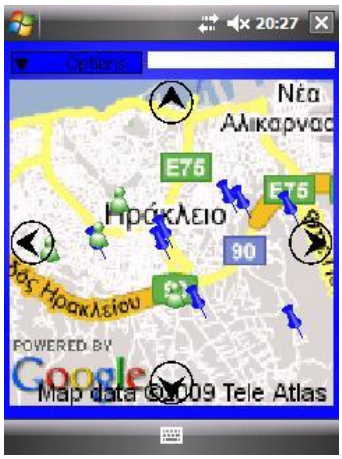


Image 5.33 Map component

As mentioned before e-kones experience offers 2 color schemes, red is for user interaction results, and blue for e-kones characteristics.

E-kones marker (business partner) this marker is placed by the application and denotes business partner. The position on the map derives from the XML that the service has sent to the application on which the business partner has before gives his geographical position to the service.

User Marker is placed by the user anywhere in the map, and denotes a custom user marker. User markers can be placed by the user either through calendar (as examined before) either through the istlMap in order to mark a specific spot on the map and add a custom note.

Ekones Users Markers are markers put by the application and denote the position of other users that are registered in the same package. Every 10000 milliseconds the application retrieves the new users locations and rearrange their positions on the map.

As previously described in order this feature to work correctly a service must be created. In this project since no service for that has implemented user positions are retrieved from a url on a server, and their locations are predefined and not produced real time.

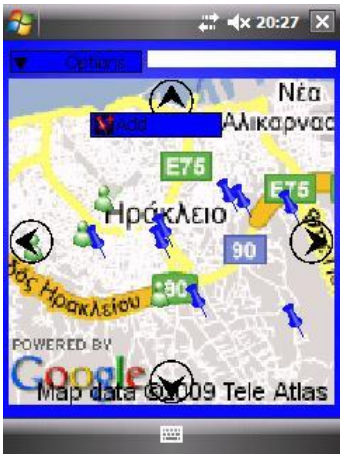


Image5.34 istlPopUp triggered by pressing on an empty map space.

Image 5.36 Business Partner istlPopUp options

Image 5.38 Distances Information Form.

Every press action on a free area on the map triggers an istlPopUp, providing the Add option (image 5.34). Selecting the add option it will create a new user marker (red color) at the pressed spot.

If the user presses on a user marker a new istlpopUp with different options this time will appear. The new options are (move, remove and Note) (image 5.35). Move option moves the marker in a new location on the map with a simple drag and drop. Remove option, removes the selected user marker from the map. Note option allows the user to add his custom note on this marker by initiating the Note Form (similar to the one described before).

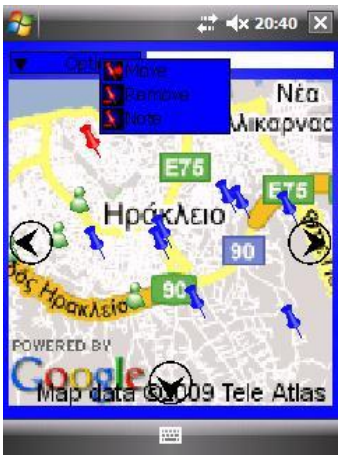


Image 5.35 user marker istlPopUp

Image 5.36 shows the istlPopUp for an e-kones user marker (Business Partner). The options offered by the istlPopUp are Business Partner Name and Distances. First option when pressed provides information for the selected business Partner by initializing the Business Partner Information Form (image 5.37). Distances options initiates the Distances the Information Form providing information about the distances between this Business Partner and the rest on the map (image 5.38)

Image 5.37 Business Partner information Form.

Pressing on a e-kones user marker the user name appears on the istlPopUp (image 5.39).

Image 5.39 e-kones users istlPopUp.

Image 5.40 istlMap component's istlMenuItem.

The last element of the istlMap component is a two level istlMenuItem named Options, which offers a variety of selections to the user (image 5.40).

First level selections are Gps Config, Notations, Users, Zoom, Ekones, Con, Exit. Each one will be examined separately.

Image 5.41 Map Configuration Form.

Con Option provides a way to force the map loading in cases where for some reason the map doesn't show. Exit option causes the end of istlMap component and returns to the main selection form. Gps Config option initiates the map configuration form (image 5.41).

Here the reader must note that options in this form haven't application impact since they removed cause of compatibility issues, instead they stay just to present the istlcombobox.

Image 5.42 Map type combobox.

The first combobox offers the ability for the user to change the requested map type. Options in this combobox are all the possible maptypes that the Static Google Maps Service offers. (Image 5.42). Second combobox offers the ability for the user to change the resolution for the requested map. Options in this combobx start from 64X64 and ends to 240X240 resolution (image 5.43). The last combobox offers to the user the ability to change the desired zoom level from 0 (no zoom) to 17 max zoom (image 5.44). Apply button submits the changes to the application while the Close button ends the Map Configuration Form.

Image 5.43 Map Resolution cobobox.

Image 5.44 map zoom level combobox

Notation Option has a second level which offers the same options as the user marker istlPopUp (image 5.45). The only new option is the Hide option which can hide / unhide all the user markers from the map.

Image 5.45 istlMenuItem Notations options.

Εικόνα 5.46 istlMenuItem επιλογή Zoom.

Image 5.46 istlMenuItem Users option.

Users option has a second level which offers the Show All and Show Friends options (image 5.46).

Zoom option has a second level which offers the Zoom In and Zoom out options (image 5.47)

Image 5.47 istlMenuItem Zoom option

Image 5.48 istlMenuItem Ekones option.

Ekones Option has a second level which offers the Hide markers option and can hide / unhide ekones markers from the map (image 5.48)

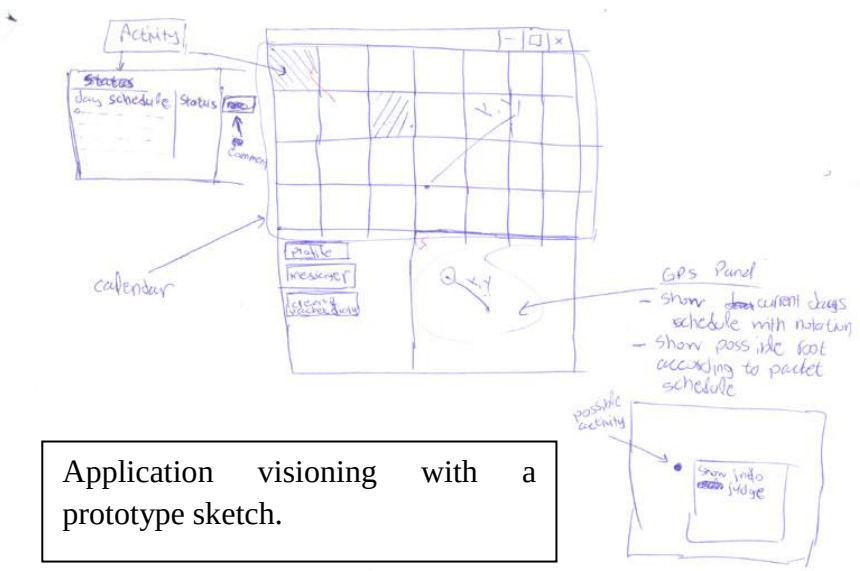
CHAPTER 6

Development Cycles and Problems.

The development of e-kones experience application last for four months, the application passed through various development cycles before the final prototype version. For start each component developed as standalone prototype, after each standalone component bound together to a single application and the main menu was build while necessary fixes were made when needed. At the same time the appropriate data classes build in order for the components to work together. After the bound was complete the XML parser along with the appropriate data classes developed in order to access the XML data from the server. At the end the interface components developed.

Along with the development miscellaneous tools developed in order to test each component (XML Parser tool, Google Maps Connection Tool, GPS tool).

During the development a number of problems arise mainly due to incompatibilty between J9 and LWUIT, some overcomed some other not and still presented in the final prototype version. Most of them analyzed in previouys pages here a sum up will be made.

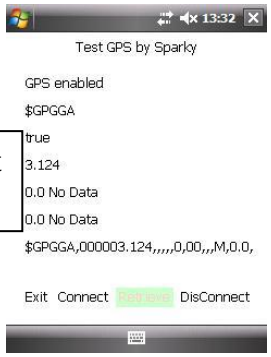


Application visioning with a prototype sketch.

1. Device denial to load google maps of a specific type, resolution and zoom level.
2. Small algorithm problems related to the convertion of degrees to screen pixels.
3. General performance problem due to the large data traffic between forms.
4. Lack of JSR-75 support from the J9, with a result not to create temp files and the performance issue to arise as metioned in previous line.

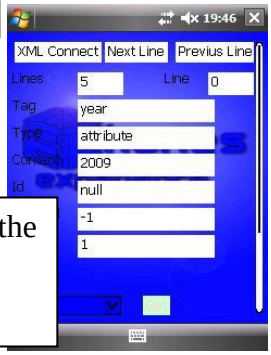
Initiate of the first development cycle: Application visioning, attempts to build the first base components.

Development of the first independent components



Development of the first critical components istlMap and Calendar

Implementation of the two critical components into the application and XML parser build.



Development of the Main Form and the auxiliary forms, tests and fixes when needed.

Development of interface components (istlpoup, istlMenuItem etc)

Final prototype version

Appendix

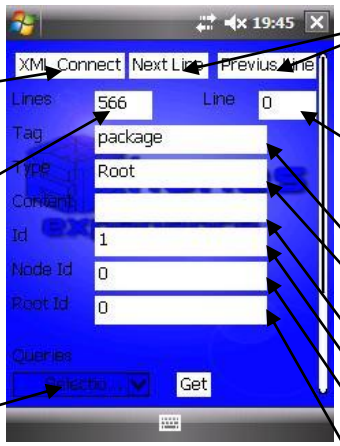
Test tool for XML Parser

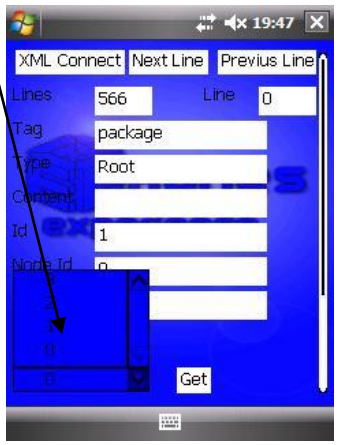
During the development of the XML parsing algorithm a test tool was created for debugging and check the parsing data purposes. The process by this tool is to retrieve a XML and put the data in order to check if the parsing algorithm was correct, after each data class was built and checked if that results were correct. Finally the XML parser along with data classes implemented to the application. Here the XML parsing tool will be explained

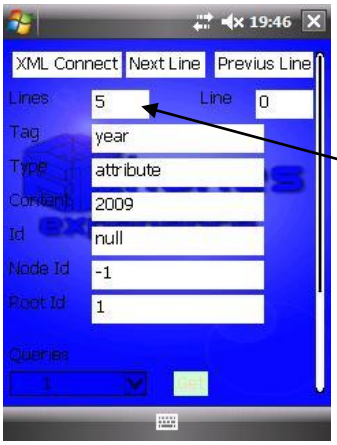
Connecting button to the internet where the XML file is.

Total number of lines read and stored into the Vector.

Way for querring the XML (1 of 4). When the connect button is pressed this istlcombobx is created with content only the XML nodes. Then with Get button a specific node tree can be retrieved.







Next / previous line (element) inside the Vector that holds the XML parsed data.

Current line

XML line information.

- Line TAG
- Type (Root, Node, attribute)
- Content (if attribute)
- Id.
- Node id
- Root id.

The node with node id 1 selected the total number of lines has changed to the number of children for this node which is 5.

GPS Tool

The GPS tool works as stand alone and displays the geographical position of the user together with some other information. The tool uses a buffer of 512 bytes to read from the com4 port, after checks if a line starting with the $GPGGA and ending with <CR><LF> is presented. If yes then cuts that line (, separated) into appropriate parts which are put in a data class that will hold the GPS data.