Using Web Services, J2ME, A* algorithm and .Net framework to construct mobile phone software applications

Daniel Drăghicescu*, Crenguţa Bogdan**

*Est Software House, Constanta, Romania; e-mail:
draghicescu_daniel@yahoo.com,
**"Ovidius" University of Constanta, Romania; e-mail:
cbogdan@univ-ovidius.ro;

Abstract

In the last years, the mobile software acquired an increasingly attention from both developers and users. The mobile applications are divided into several categories: games, multimedia, and utility applications. In the last category, we can place the applications for guidance in a city, which offer services like path finding, area information, and so on. Today, these applications are mainly developed using the GPS technology. The main objective of our study was to use different software technologies, like web services, J2ME, heuristic algorithms, and .NET framework to develop a complex mobile application. This paper presents the Mobile Path Finder (MPF) application for the guidance of the user in a city. The application uses all the above technologies and the A* algorithm to allow a mobile phone user to find the optimal path between two intersections of a city. MPF is a distributed and n-tier application, divided into four subsystems: server, administration, client, and data storing. Each one provides different functionalities needed for the application operating. MPF is based on the client-server architecture. The client subsystem runs on a mobile phone, but all of the data processing is done remotely on a dedicated server, in order to bypass the phones small computing power. The server, administration and persistence subsystems run on one or more regular PC's.

1 Introduction

In this paper, we give a solution to a problem that affects people more and more each day: people's mobility. It doesn't matter if we are talking about professional interests or personal ones, we often find ourselves in the position to adapt in a newly, not-known environment. Even if we are talking about a business meeting or an unexpected trip with our family, the outcome is always the same: we have to learn quickly a foreign area or a new city.

This problem has been in our attention for a long time. Today, we gave up the traditional maps and took the next step thanks to the technology that

we have created. Starting with maps covering the entire Earth and ending with the one that reach even the street level, all the important internet corporations offers online maps to aid the users to learn about the areas with which they are not familiar.

Another proof for the importance of this subject is given by the technology developed by the Unites States Department of Defense, the Global Positioning System (GPS) [14]. Using 24 satellites that orbit around the Earth, this technology allows a GPS Receiver to determine its exact global position, velocity, and direction. With this data attached to a local map, the positioning problem is fully solved.

1.1 The Solution

The technology's evolution has always leaded to human progress. If a couple of years ago we tried to make technology go faster, in the present, we try to make it smaller. Because of the peoples need for mobility, we are living in a new era for the mobile devices. There are a lot of well-known mobile products available, like laptops, but only one excels in popularity: the mobile phone. The main advantage of a mobile phone is its small size, making it very easy to carry around and work with. Nevertheless, this type of mobile devices has a great disadvantage: its small processing power.

The purpose of this paper was to develop an application that runs on a mobile phone and makes the most of its processing power, without being affected by the phones small processor and memory. The application should aid a user to find the path between two points, which represent intersections of a city. We did not want to use the existent GPS technology; therefore, we try to offer an alternative for it. Our solution uses the actual software technologies that are briefly presented in the next section.

2 Used Technologies

2.1 Web Services

Web services are the fundamental building blocks in the movement to distributed computing on the Internet. Open standards and the focus on communication and collaboration between people and applications have created an environment where web services become the platform for application integration.

There are some definitions for the web services. During our research, we considered the web service as a software system identified by an URL address

whose interface and public binding are defined and are described using XML [4].

The main advantage of a web service architecture is interoperability, that it allows programs written in different programming languages, running on different platforms, to communicate between them using a certain standard. If this promise was given in the past by CORBA [3], the SOAP architecture [15] brings a more simplified approach, lowering the barrier for SOAP standard implementation. SOAP implementation is sustained by software corporations and independent developers.

Another advantage is that web services work with standard Web protocols - HTTP, XML, and TCP/IP. Most companies already own web infrastructures, so the cost for adopting web services technology is much lower than similar technologies [13].

2.2 Java 2, Micro Edition

Java 2, Micro Edition (J2ME) is the Java edition for limited hardware devices, like mobile phones, embedded devices and other handheld devices [7]. J2ME is aimed at devices with a small amount of memory (more precisely, 128 KB) and processors much slower than the ones used for a regular desktop computer.

J2ME does not define a new programming language, but it adapts the existing Java technology [6] for the mobile and embedded devices. To address the stricter limitations of small devices, J2ME sometimes replaces the standard J2SE API and adds new interfaces, but the evolution for J2ME stands mostly in omitting unnecessary parts in J2SE and J2EE and adding new ones designed for mobile devices.

2.3 A* Algorithm

A* (or A-star) is a general search algorithm that is extremely competitive with other search algorithms. The search algorithms are used in a wide variety of contexts, like artificial intelligence [12] or sentence parsing. That is why a good search algorithm should allow us to solve a large number of problems with greater ease.

A* algorithm is best used in space orientation problems. In every space orientation problem, we meet two aspects: the initial conditions defining an initial state and the purpose of the problem as the final state. For every action we trigger, we generate all of the successor's states to represent the action's effect. If we keep on repeating this last step, at a certain point the successor will be the same as the final state, and the path from the initial state to the

final one will represent the problem's solution.

A* generates and processes the successors in a certain order. Each time it looks for the next step, the algorithm uses a heuristic function to try to choose the best solution. If the heuristic function is optimal, A* will find the result in a very short time.

The algorithm uses two lists, which are called Open and Closed. The Open list will contain all the nodes that should be examined and the Closed list the ones that have already been examined. Initially, the Open list contains the starting node and the Close list is empty. For each node, we have to keep the results of these functions (see Table 1). Each node keeps a reference to

Table 1
The definition of the function f, g, and h

g(n)	the total cost from the initial node to n
h(n)	the estimation, according to the heuristic function, of the cost needed to reach the final node from n
f(n)	=g(n) + h(n); intuitively, this is the best solution that passes thru n

its parent, so we could know what the path is after we have reached the final node.

A* has a main loop where it reads the node n with the smallest value of f(n) from the Open list (in other words, the node it thinks it is included in the optimal solution). If n is the final node, the solution has been found and it can be returned by reading each of the node's parents, starting from the end. Otherwise, n is removed from the Open list and added in the Closed list. The next step is to generate all of n's successors.

For each successor n', if it is already found in the Closed list and has an initial value for estimated f smaller or equal then the calculated value, the algorithm drops n' and moves on (this means that n' was already used, because it is in the Closed list). Similarly, if n' is found in the Open list and has an estimated f value smaller or equal then the calculated value, the algorithm drops n' and moves on.

If we can not find a better version for n' in the Open or Closed lists, we remove n from the two lists and set n as a parent for n'. Also, we have to calculate the estimated cost for n':

- g(n') = g(n) + the cost for reaching n' from n
- h(n') = the heuristic estimation from reaching the final node from n'

- f(n') = g(n') + h(n')

Finally, n' is added in the Open list and the loop is repeated.

A* success depends very much on the heuristic function chosen. For each problem, we can find bad functions or good functions. A good function will allow the algorithm to find the optimal solution is a short time interval, while a bad function could raise the execution time or even force the algorithm to return a wrong solution.

To guarantee that we will find the optimal solution, if one exists, the heuristic must be "admissible". To be admissible, a heuristic must never overestimate the cost of getting from a state to the goal state. Another issue is how quickly the heuristic function can be computed. A perfect heuristic is almost never available, and to come close requires significant additional computation. A lot of the time, a heuristic that only comes close to a perfect estimate, but runs very quickly, is superior to one that is perfect but takes forever to do so.

3 Mobile Path Finder software application

In this section, we present the most important steps from the developing process [11] of the MPF software application that offers the solution to the problem presented in the beginning of Section 1.

3.1 MPF software analysis

During the software analysis of the MPF system, we identify the software actors of the systems, i.e. the roles of the users of the system and the system behavior in its interactions with the software actors [11].

With the help of MPF software application, a user will be able to find the path between an intersection inside a city and another one, from the same city. More precisely, we call a *path* a street succession inside a city that allows a person to travel from one intersection between two streets to another one. The first intersection is called the *source* and the second one is the *destination.*

The MPF application has two possible user roles. The *user* uses only the application running on the mobile phone. He or she cannot use any other functionalities of the application and the information he/she sends or receives is related only to the path needed inside a city. He or she will authenticate as a valid user using as credentials a username and a password given by one of the system's administrators. The user may specify the city he/she is located in, its current position, and destination, and the system returns the path he/she needs to follow.

The *administrator* uses the system to manage its functionality. His or her responsibility is to deal with the user accounts and with all the information about the available cities, as well as all the streets inside them.

3.2 MPF software design

MPF is an n-tier, distributed application, designed using the client-server architectural style [11]. The client-side runs on a mobile phone, but all of the processing is remotely done on a dedicated server. In this way, we bypass the phone's small processing power. The server-side runs on a regular PC, the same as the administration module.

At the architectural design level, the application is structured in four different modules:

- The *client module* runs on the user's mobile phone and achieves the application's main purpose, to help the user find his or hers path inside a city.
- The *server module* runs transparently for the user; its purpose is to process all the data from the client module and return the responses to the user.
- The *administration module* runs on one or more computers and allows an administrator to interact directly with the system in order to manage its functionality.
- The *database module* stores all of the persistent data needed for the system to run properly.

3.3 MPF implementation

In this section, we will take into consideration the MPF implementation methods. We do not focus on ease operations like adding a new user or changing the name for a city, instead we present the more interesting methods used for every module of this application.

3.4 The administration module

The biggest challenge for this module is the way we want our administration process to work. We want a module that can handle, beside the users, any number of cities, each one with important information, like every street and intersection belonging to the city. In addition, the administration should be

an unsophisticated process, so any user can used it, regardless of his computer experience.

Since the visual experience is by far the simplest understanding process, we want that the administrator to be able to load into the administration module maps for the cities and define streets and intersections, as well as other information, like street length or name. The maps will be loaded from picture files, like jpeg's or gif's, and the user will actually work "on them", specifying to the system where is each intersection and street located. More than that, the user will actually have the possibility to set easily all the necessary information only by double-clicking on the street or intersection he wants.

The main task for this problem is creating a user control for loading large picture files at maximum resolution and interacting with them. Since we want our control to work at an optimum efficiency, we load the entire map, but only draw the items that are currently on screen. In this way, we process the data only for a section of the map, and not the entire map.

The control is a panel, i.e. object of a class that extends the Panel class located in System.Windows.Forms. Because maps are usually large files, we can expect that the entire map does not fit on screen, so scrollbars are needed. We add to this new class the possibility to specify an Image object to load, which represent the object created from our map file, and we override the On-Paint event to load the portion of the map that should be displayed on screen, as well as the streets and intersections from that section. To find out which section of the map should be drawn, we first get a rectangle containing the section currently on screen, related to the scrollbars position, and display from the Image object only that rectangle. We represent the intersections as filled

Table 2
An excerpt from the OnPaint method

```
protected override void OnPaint(PaintEventArgs e){
...
Rectangle src = new Rectangle(-AutoScrollPosition.X,
-AutoScrollPosition.Y, this.Width, this.Height);
Rectangle dest = new Rectangle(0, 0, this.Width, this.Height);
e.Graphics.DrawImage(imagine, dest, src, GraphicsUnit.Pixel);
...
}
```

circles and the streets as parallelograms. Finding out which ones should be visible at a certain time resumes to a simple coordinates compare, but only if we remember to always add the current scrollbars position, AutoScrollPosition.X

and AutoScrollPosition.Y.

Let us take a closer look at the way we draw the streets on the map. To define a street section, the user must initially click the two intersections that represent the beginning and ending points for the street. To draw a street segment we draw a parallelogram, so we have to find out all of the four points coordinates. Since we do not know in which order the user will select the two intersections, we have to sort them before we can create the street segment. The sorting will be done from "left to right", or in other words, the street section begins with the left intersection and ends with the right one. If the two intersections have the same vertical coordinates, we will arrange them from "top to bottom", defining the initial intersection as the upper one and the final intersection as the lower one.

Using this sorting method, we are sure that the streets are always represented as shown in Figure 1. The points A and D will always be inside

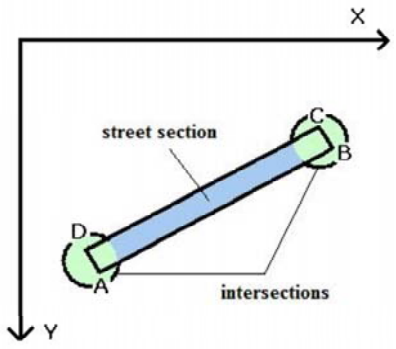

Figure 1. Streets and intersections representation

the intersection that represents the beginning of the street and the points B and C will be inside the intersection defining the ending of the street. Now we have to assure that we calculate correctly the coordinates for the points A, B, C, and D. Let (X_b, Y_b) be the center for the beginning intersection, D_b its diameter, and (X_e, Y_e) the center for the ending intersection and D_e its diameter.

Now we have two possibilities. If the beginning intersection is "below" the ending intersection, or in other words $Y_b \leq Y_e$, the point's coordinates are:
$A(X_b + D_b/4, Y_b + D_b/4)$
$B(X_e + D_e/4, Y_e + D_e/4)$
$C(X_e - D_e/4, Y_e - D_e/4)$
$D(X_e - D_b/4, Y_b - D_b/4)$

If the beginning intersection is "above" the ending intersection ($Y_b < Y_e$), the coordinates will be:
$A(X_b - D_b/4, Y_b + D_b/4)$
$B(X_e - D_e/4, Y_e + D_e/4)$
$C(X_e + D_e/4, Y_e - D_e/4)$
$D(X_b + D_b/4, Y_b - D_b/4)$
Now that we have our streets and intersections on screen, we have to find a way to pick the one the user double clicks on an intersection. To verify if an intersection is simple, we just have to determine if the point where the double clicking occurred verifies the equation for a point inside a circle: $(x - x0)^2 + (y - y0)^2 R^2 < 0$, where (x, y) are the mouse coordinates for the double clicking, $(x0, y0)$ the intersection center and R its radius.

For a street, we have to use a computational geometry method to find out if a point is on the inside or on the outside of a convex polygon. Let A, B, C be three points. The area for the triangle ABC is:

$$A = \left\| \frac{1}{2} \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix} \right\| \quad (1)$$

If the points are in trigonometrically order, then the value for the determinant from the formula will be positive, and the triangle will be called "positively defined". If the points are in reverse trigonometrically order, the value is negative, and the triangle is called "negatively defined". If we use the same notation for the points as shown in Figure 1 and remember that the center of the axis is on the monitor's top left corner, then the points A, B, C and D are always in a reverse trigonometrically order. A point M will be inside the polygon if the triangles MAB, MBC, MCD and MDA are negatively defined, or in other words all of the four determinants for each three points have a negative value.

At the implementation level, the administration module is a stand-alone GUI Windows application, written in C# [5], under .NET Framework 2.0 [10]. The main graphical user interface of the administration module is presented in Figure 2.

3.5 The server module

The server module is an ASP .NET 2.0 Web Service [1], written in C#, running on a dedicated computer, under a HTTP server (for example, IIS Internet Information Services, included in Windows XP/Vista). The communication

Figure 2. The main graphical user interface of the administration module

between the client and the server modules takes place over HTTP using the SOAP protocol [15].

The responsibility of the server module is to get the client's requests, process them, and return the answer back to the client. This way, the module does all the calculations and works with the database module.

Let us take an example to see how this module works. Let us say the user want to pinpoint his current location. From his mobile phone, he inputs the name for the two streets that form the closest intersection from his current position. When choosing to continue, the client module running on the mobile phone uses a wireless Internet connection to contact the server module, or in other words, the web service. The client passes the inputted values to the web service and waits for a response. The web service contacts the database module and gets all the valid streets that contain the inputted values and return to the client two lists, each one containing valid street names. Using this technique we can be sure that all of the processing will be done remotely by the web service, thus bypassing the phone's small processing power.

The most important operation done by the server module is probably the finding of the best path between the two intersections inputted by the user. The first thing to do after establishing the source and the destination intersections is to get from the database all of the streets and intersections for the city we are interested in. For a city, the streets and intersections actually

form an undirected graph, with the intersections being vertices inside the graph and the streets edges between vertices. The graph is undirected because inside a city there is no distinction between the two intersections associated with a street section, as we can cover a street in both directions.

Let us take an example with a section from a city with five intersections (annotated from A to E) and five street sections between them, as shown in Figure 3. Each street section has information like name and length. All this

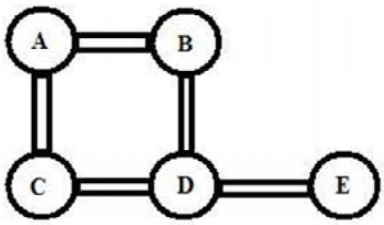

Figure 3. Graph example

information has been inputted by an administrator using the administration module and saved into the database module. In the tables 3 and 4, we show how the intersections and streets were saved into the database (that runs under Microsoft SQL Server 2005 [9]).

Table 3
Intersections table

IntersectionId	Name	XCoordinate	YCoordinate	City
1	A	2449	2386	1
2	B	2477	2375	1
3	C	2504	2368	1
4	D	2530	2357	1
5	E	2557	2348	1

For an intersection, we have a unique ID generated by the database, an optional name, the X and Y coordinates used to represent graphically with a circle the intersection inside the administration module and the city for the intersection.

For a street segment, we are interested in information like a unique ID, the name and the street section's length (in meters) and the beginning and the ending intersections, taken from Table 3. We can also add other information, like the coordinates needed to draw the streets inside the administration module, but to make the example simpler we resume only to what we need.

Table 4
Streets table

StreetId	Name	Length	BeginIntersection	EndIntersection
1	Street AB	40	1	2
2	Street AC	50	1	3
3	Street CDE	40	3	4
4	Street BD	50	2	4
5	Street CDE	40	4	5

We do not need to store the city for a street segment, because we already have it from Table 3. We have to remember that we are storing street segments, and not an entire street. A single street contains more than one intersection and we need to save each segment between two consecutive intersections. In this example, Street CDE is a single street, but has two segments, C-D and D-E.

With this information organized, we create a Graph object using the street segments as edges and the intersections as vertices. The metric used will be the street segments length. It is very important to remember to create the edges in both directions, because our graph is undirected. For example, for the first street (Street AB) we need to create an edge between vertices 1 and 2 and another edge between vertices 2 and 1.

Having the entire graph in memory, we run A* algorithm on it and find the best path between two intersections.

3.6 The Client Module

Because of the way we designed this system, the client's module task is very simple. Its responsibility is to get the input from the user, pass it to the server module, and display the response it receives. In addition, the changes made to the system do not affect the client module, because the server module is the one doing all of the processing.

The client module is implemented in Java, using Java 2, Micro Edition platform [7]. For the application to run, the mobile phone has to have the J2ME virtual machine, as well as the Web Service API (JSR 172) [8]. In addition, it needs a GRPS/3G internet connection or another type of wireless internet connectivity. The graphical user interfaces of the client module are presented in Figure 4.

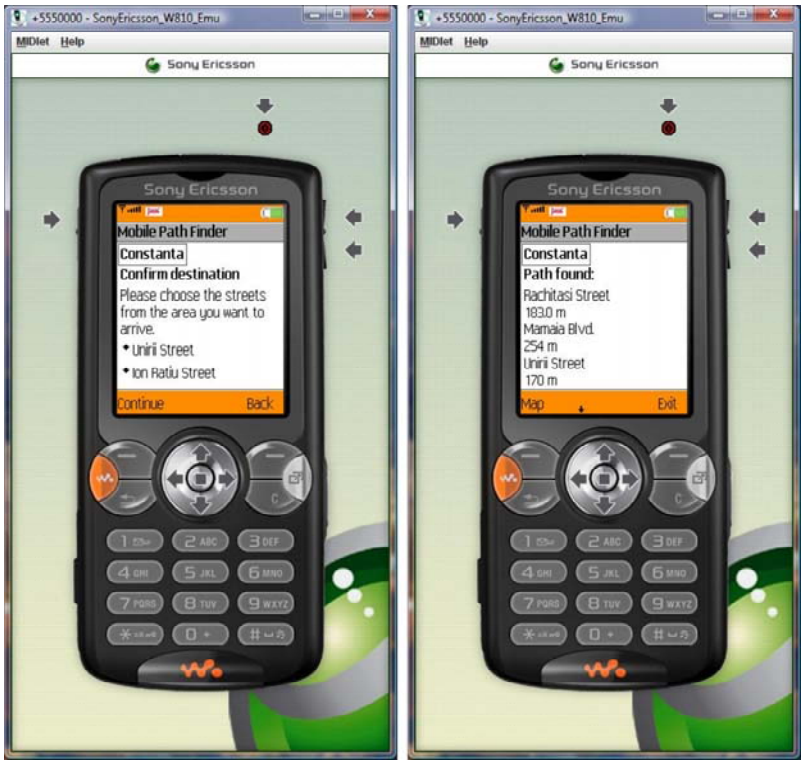

Figure 4. The graphical user interfaces of the client module

4 Conclusions

This paper presented how the latest technology combined with some mathematical knowledge can help us create new software applications to lower the barrier between people and the common problems that we face every day.

In this respect, we developed the MPF application for the guidance of the user in a city. The application uses different technologies like web services, J2ME, heuristic algorithms, .NET framework and the A* algorithm to allow a mobile phone user to find the optimal path between two intersections of a city.

From an architectural point of view, MPF is a distributed and n-tier application, divided into four subsystems: server, administration, client, and data storing. Each one provides different functionalities needed for the application operating.

In addition, MPF is based on the client-server architecture. The client

subsystem runs on a mobile phone, but all of the data processing is done remotely on a dedicated server, in order to bypass the phones small computing power. The server, administration and persistence subsystems run on one or more regular PC's.

If the GPS technology is expensive and impossible to develop from scratch, the solution provided by MPF is available at a low-cost for every user and it is easy to control by any developer that wants to implement it.

References

[1] ASP .NET 2.0 Web Services, http://msdn2.microsoft.com/en-us/webservices/bb245930.aspx

[2] A* Algorithm, http://the.jhu.edu/upe/2003/01/18/introduction-to-the-a-star-algorithm

[3] Common Object Request Broker Archietcture (CORBA), http://www.omg.org/corba-e/index.htm

[4] Extensible Markup Language (XML), http://www.w3.org/XML/

[5] Hejlsberg, A., Wiltamuth, S., Golde, P., *The C# Programming Language*, Addison-Wesley Professional, 2004.

[6] Java Platform, Standard Edition, http://java.sun.com/javase/

[7] Java 2, Micro Edition, http://developers.sun.com/mobility/apis/articles/wsa/index.html http://developers.sun.com/mobility/getstart/articles/survey/

[8] J2ME Web Services API, JSR 172, http://java.sun.com/products/wsa/

[9] Microsoft SQL Server 2005, http://www.microsoft.com/sql/default.mspx

[10] .NET framework, http://msdn2.microsoft.com/en-us/netframework/default.aspx

[11] Pressman, R. S. , *Software Engineering. A Practitioner's Approach*, McGraw-Hill Publishing Company, 2000.

[12] Russel, S., Norvig, P., *Artificial Intelligence. A Modern Approach*, Prentice Hall, 1995.

[13] Web Services, http://msdn2.microsoft.com/en-us/library/ms996507.aspx

[14] Wells, D., Kleusberg, A., *GPS: A multipurpose system*, GPS World, **1(1)**(1990), 60–63.

[15] World Wide Web Consortium, *SOAP Protocol*, W3C Recommendation, http://www.w3.org/TR/soap/