The ProxyGator – Server-side Wireless Toolkit

Dushiant Kochhar and Abdelsalam (Sumi) Helal

Computer & Information Science & Engineering Department
University of Florida, Gainesville, FL 32611, USA

helal@cise.ufl.edu

ABSTRACT

Mobile devices have characteristics that are totally different from desktop environment - small displays, low processor power, less memory and low bandwidth of the link. New platforms, like Java 2 Micro Edition, have been developed for such types of devices, which provide limited capabilities to the mobile client. For better user experience, most of the mobile applications on the J2ME platform make use of a proxy, which sits on fixed infrastructure and acts as an interface between the client and the server and does a lot of the work on behalf of the mobile clients. Developing applications requires a lot of effort in terms of time, and developing proxy along with client module may increase the development time significantly.

To reduce the development time, we are introducing the *ProxyGator* toolkit. It is a server-side wireless toolkit, which includes an API that provides common functionalities used in a typical proxy, like generating HTTP requests to different websites, parsing the HTML data received from websites, sending the datagram packets, converting the images from any format to PNG format, connecting to the databases, string manipulation and searching the UDDI registry. Also, to make things easy for the developer, a toolbar is provided which provides help on the API, generates code for common functionalities used in a proxy and also provides help on WSDL documentation and UDDI search for the Web Services. Toolkit also includes a wizard, which generates a proxy shell according to the options selected by the developer. This whole toolkit infrastructure is built within Visual C++ 6.0 IDE. We present the ProxyGator Toolkit and give an analysis to show the saving in development time achieved by using this toolkit.

1. INTRODUCTION

Mobile computing is becoming ubiquitous these days with the advent of catchy devices and new applications. But mobile computing environment is handicapped by a number of factors, like:

- Less processor power
- Less memory
- Small and varying screen sizes depending upon the devices
- Less and variable link bandwidth

The above factors force the mobile applications to be small and efficient, consuming less processor power and having small memory footprint. The applications should be able to work in low bandwidth environment. In addition, the mobile applications should be designed in such a way so as to adapt themselves to the changing user interface and bandwidth availability.

This has lead to the development of new platforms targeted specifically for mobile clients. One such platform targeted towards the development of mobile applications on mobile devices is Java 2 Micro Edition or J2ME. It provides just the essential capabilities to the mobile device through its configuration of JVM and APIs.

Due to limitations of mobile platforms due to inherent limitations of mobile devices and, added requirements on part of the mobile applications for adaptation, applications for such an environment need to have a proxy, sitting between the client and the server that does most of the work for the client. The placement of the proxy is shown in Figure 1. A typical proxy has two interfaces - a client interface and a server interface. The client interface interacts with the client and receives requests as well as sends the responses. This client interface usually uses very limited number of protocols (e.g., UDP) due to limitations of the client. The server interface attaches to different servers for fetching data and uses a number of protocols like TCP, UDP or SOAP etc. The third module is Adaptation Module, which transforms data according to the needs of the client. This helps in reducing burden not only on the client but also on the server, which, in many cases, is feeding both mobile as well desktop clients.

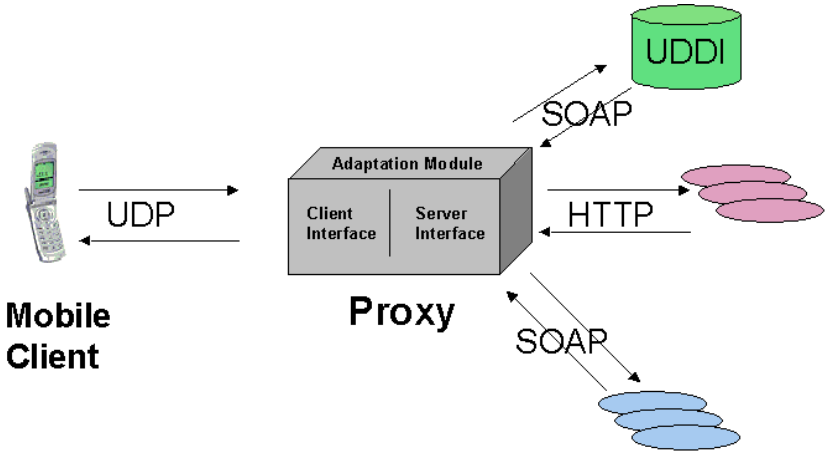

Figure 1 - Proxy - with respect to other components in wireless environment

This approach of having a proxy in between the client and the server is used in most of the mobile applications designed on J2ME platform. So, when developing a mobile application, the developer needs to develop not only a client module but also a proxy to cater to the needs of a resource-crunched client, assuming that the server is already developed. The development of a proxy demands the same amount of resources as development of a client module or even more, in an already tight development period. Due to the importance of a proxy, good development tools need to be provided to a developer for its quick and efficient proxy development.

Although there are a number of development tools available for client-side development on J2ME platform, such as J2ME Wireless Toolkit from Sun Microsystems and CodeWarrior from Metrowerks, there is no integrated development environment available that caters to the rapid development of the proxy on this platform. In other words, a developer himself has to collect all the components required for the proxy development. The proxy is an important part of any application as is the client module in mobile environment. So, the way it is developed, adds to the overall success of an application, as it directly affects what features can be included in the application and what cannot. Even if a mobile client is limited in resources, a mobile application can still compete with a desktop application if the proxy is intelligent and does a lot of work in an efficient way on behalf of both the client and the server.

This proxy in its primitive form just forwards data from the server to the client while an advanced version of it may adapt the output stream to the client that takes care of the client's capabilities and link

bandwidth [1, 2]. For later cases, the proxy needs to be intelligent enough to keep track of client's current position and device. This way it can enhance user experience.

2. GOALS

The goal of our work is to provide complete infrastructure to the developer so that a proxy can be developed rapidly. In other words, it is to develop a server side toolkit for wireless environment that would complement client side toolkit for J2ME platform. For this, we have developed ProxyGator, a toolkit that contains an easy-to-use API and a host of other features. This API provides support for the most common functionalities such as parsing the data, maintaining user profile data in database, providing support for HTTP, etc. ProxyGator emphasizes some of the following things:

- Includes common functionality in the form of an API.
- Provides help to the developer on how to use the API.
- Minimizes coding required by the developer by providing several code generators such as the one that generates generic proxy shell.
- Provides complete integrated development environment to the developer, which should include an editor, a debugger and a host of other features provided by today's Integrated Development Environments or IDEs. To cater to this requirement, the toolkit is developed within Visual Studio 6.0 IDE.
- Provides infrastructure for Web Services, which enables programs to discover each other and form an application.

3. TOOLKIT COMPONENTS

The *ProxyGator* toolkit has not only been developed in VC++ 6.0 but also been integrated into the Visual C++ IDE 6.0. Visual C++ and its IDE have been chosen due to a number of reasons.

1. Visual C++ IDE is a complete integrated environment containing an editor, a debugger, a compiler and a host of other features.

2. Visual C++ provides a lot of support in the areas of development of toolbars and custom wizards, which can be integrated into Visual C++ IDE itself.

3. Microsoft foundation classes, part of VC++, provide a rich set of APIs for software development.

4. There are numerous other components being available on Visual C++ along with a large developer base of Visual Studio that will further help the mobile application developer in development.

Toolkit consists of three components:

- API - The API provides a number of interfaces for providing commonly used functionalities.
- Toolbar - Named *MobileToolbar*, it is an add-in for Visual C++ IDE 6.0. It provides support to the developer in the form of API help, searching the UDDI, going through local as well as remote WSDL documents and code insertion.
- Wizard - Named *MobileProxy*, it is a Custom Appwizard in Visual C++ IDE 6.0, which helps in creating a shell for a proxy depending upon the options selected by the developer.

3.1. API

3.1.1. Requirements

As part of the mobile computing class of spring 2001, students at the University of Florida developed a number of applications on J2ME platform supported by Motorola iDen group's i85 phone [3]. The

applications that were developed in the class addressed whole spectrum of applications for phones, like providing weather service, airline reservation system, stock management, travel expense management, shopping assistance, providing directions on phones etc.

We studied those applications from the point of view of what should be included in the infrastructure as ready-to-use interfaces for the developer. These interfaces should provide the most commonly used functionalities in a typical proxy and make it easy for the developer to write a proxy.

3.1.2. Support Included in API

Not surprisingly, all the applications developed, made use of a proxy. After studying those applications, we short-listed the API sets, which need to be included in the toolkit, for commonly used functionalities in those proxies. The use of these APIs will greatly reduce the development time as the API methods directly address the requirements of writing a proxy. Following are the API sets, which we have included in our toolkit:

- Text Extraction from HTML data – There usually is a need to extract relevant information from the data coming from the servers. The data in most cases comes from different websites, which make use of HTML format. We have provided the API that lets the developer extract the information between tags and giving start as well as end position of extraction.
- Sending datagram packets – This API provides interfaces to send the datagram packet to required destination and port and also fragments the packet according to developer's requirements.
- Generating Http Requests - This API can fetch the response from a server making use of Http protocol. Different options to fetch the data at periodic intervals, either in a string or in a file, are provided.
- Image Conversion – As the Motorola iDen's i85 phones, running on the J2ME platform are capable of handling only PNG format; we have included the API that provides methods to convert images from any format to PNG format. This also provides options to change size, resolution, and rotation of the image. This makes use of 2PNG software [4].
- User Profile maintenance - Through this API, the developer is given all the methods to maintain user profile consisting of name, address and properties of the client. Methods that are included in this API set are - select, update, delete and insert.
- String Manipulation – This API provides support for encoding data through the use of delimiters and also extracting data out of a string that is encoded for sending or receiving from or to the clients.

3.2. WEB SERVICES AND THE TOOLKIT

Web services, based on XML, are the basic building blocks in a move towards distributed computing on the Internet [5]. Web services enable applications to be constructed from different blocks regardless of where they reside, the platform they are running on and the way they are implemented. For this, Web services provide infrastructure and protocols for one block to discover other necessary blocks residing on the Internet and communicate among them. Web Services define how to describe interfaces for the programs written in such a detail that other programs can invoke it. Web Services is a big shift as it enables programs to discover each other and hence form complete applications.

Web Services has three main components:

- SOAP [5] - For different entities to communicate in Web services, Simple Object Access Protocol or SOAP is used. It is based on XML and consists of SOAP envelope containing header and body.
- WSDL [5] - Web Services Description Language or WSDL is XML based language, which is used to define Web Services and describe how to access them. It gives all the technical details on accessing the service like, what are the methods provided in the service, their input and output parameters and their data types.

- UDDI [5] - Universal Discovery Description and Integration or UDDI is a registry, which publishes the services provided by the service providers, and also enables search on those services just like Google [6]. Through its inquiry API it helps in discovery of the services.

3.2.1. UDDI Support in Toolkit

Since Web services are rapidly gaining popularity, more and more mobile applications would like to make use of Web services world. Again, limitations of mobile clients make it difficult to inquire deep into the UDDI registry, connect to different access points and communicate using SOAP. And again, a good solution is to hand over this responsibility to the proxy, which drills down the UDDI and then connects to the access point and fetches the data.

To search a UDDI registry, a proxy needs standard inquiry API as mentioned in the UDDI specifications [7]. To accommodate this need, the ProxyGator includes an API to search the UDDI registry. We have provided both find_xx as well as get_xx API classes to inquire UDDI on the basis of businessEntity, businessService, bindingTemplate and tModel. For each class, different methods are provided to add identifiers and categories in the search criteria. With this API, the proxy can drill down in the UDDI registry and find the service it is looking for. The proxy program can be developed such that this drilling can be done at run time.

The developer of a proxy might be looking for particular service based upon on a keyword or a tModel and then write the proxy accordingly. API support includes a class *getServices*, which has the methods to search the UDDI registry based upon a keyword or a tModel that returns up to 100 results containing all the information about suitable services. The result contains the following information:

- Business Name - The name of business providing the service.
- Business Key - The business key identifying that business name.
- Service Name - The name of service (e.g., Weather service.) matching the keyword or tModel.
- Service Key - The key given to that service which uniquely identifies it.
- Access Point - The point where that service can be accessed (e.g., access point can be http://www.learnxmlws.com/services/weatherretriever.asmx).
- Description - This contains description of the service.
- Binding Key - Binding key for that service.
- tModelKey - tModel that uniquely identifies that service.
- OverviewURL - This contains URL providing information on how to access the service and in most cases it points to URL from where WSDL document can be accessed.

The usage of this API will lead to less dependence on the standard drill down search pattern of UDDI and hence result in lesser development time.

Along with this API, the developer can make use of GUI support, included in *MobileToolbar*, for searching the UDDI registry, as explained Section 3.3.

3.2.2. WSDL Support in Toolkit

The importance of WSDL in the overall picture of Web services is greatly enhanced if standard WSDL documents are defined for specific industry [9]. If interfaces are standardized through WSDL and service providers adhere to them, then the service users can switch from a vendor to another if one vendor stops providing service and, their programs will still work. Working on this basis, we have come up with twelve services, which are most commonly used in our day-to-day lives and, we have provided WSDL documents for these twelve services.

- Airline - Its WSDL document provides detail on what are the operations involved in Airline reservation system, the messages exchanged and their parameters.
- Movies - For service developers and users of this service, WSDL document of this service gives operations for queries about movie releases.
- News - Its WSDL document gives details on how news should be implemented and accessed.
- Stock - Its WSDL document gives details on how service related to stock queries should be implemented and accessed.
- Travel - Its WSDL document gives details on travel information services like hotel reservation and room availability.
- Weather - Service providers and users of weather service will look into its WSDL document.
- Yellow Pages - This service is similar to yellow pages we use daily and its WSDL document explains its technical specification.
- Sports - Sports related service should have interface as described in its WSDL document.
- Shopping - Different messages and operations involved in shopping are described in its WSDL document.
- Medical - Interface for medical services, like doctor availability and scheduling an appointment, is described in its document.
- Banking - Banking service should expose the interface described in its document.
- Bidding - Online bidding service should have interface as described in its WSDL document.

The idea behind providing WSDL documents for different services is that they can be registered as tModels with a UDDI registry and vendors interested in mobilizing their services will adhere to these tModels. On the other hand, the application developers can search UDDI on basis of these tModels. One step further, program can be developed in such a way that if one access point for the service is not working the program automatically looks for a different access point.

3.3. MOBILETOOLBAR

MobileToolbar, a toolbar add-in for Visual C++ IDE 6.0, is the second component of the toolkit. It is shown in Figure 2. Labels on the figure show all the buttons on the toolbar, whose functions are explained next.

1. API Methods - This is the first button on the toolbar and clicking it displays the set of APIs we have provided along with the classes and the methods in each of them. Whenever developer selects any method, code for call to that method is inserted at the current cursor position in the editor. We have tried to give names to the parameters taken by each method and their data types. This way developer knows what arguments to pass and in which order during the function call.

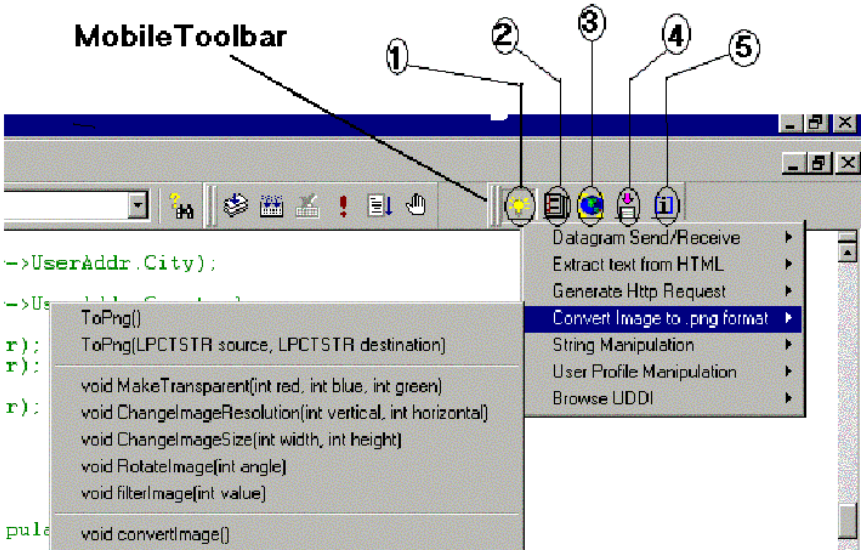

Figure 2 - MobileToolbar within Visual C++ IDE 6.0

2. WSDL documents - This is the second button on the toolbar, which helps in coding of the proxy by giving the details on how to access a service. This is done by providing WSDL documents, both local as well as remote. It provides two options to the developer:

(a) Look for local WSDL documents - In this case, names of all services appear whose WSDL documents we have come up with, as explained in section 3.2.2. On selecting one, document is displayed on the Macro window of the output window in the IDE.

(b) Look for Remote WSDL documents - With this option, developer has to give complete website address where that WSDL document lies. The document is fetched from source and again displayed on the Macro window of the output window in IDE.

Displaying the document within the IDE is advantageous since the developer can refer to it whenever he wants to while writing the program.

3. UDDI Search - This is the third button, which provides dialog box for UDDI search. The developers can search on the basis of either:

(a) Keyword - With this option, all the services whose names contain the keyword mentioned are returned from UDDI along with the complete details of businesses providing those services, service access points and overview URLs, which provide URL for information on how to use the service. In most cases, overviewURL points to the location from where the WSDL document can be fetched. The developer can further specify whether he wants to search for only those services whose technical details are given by the WSDL document.

(b) or, tModel - In this case, the developer has to mention the tModel for the service he is looking for. The search is made for all the services, which contain that particular tModel in their tModelBag. Again, output is displayed in Macro tab of IDE's output window.

4. Insert Code - This button provides options for inserting the code corresponding to different functionalities, at the current cursor position in the editor. The options included here are selected taking into consideration the functionality that is most commonly used in the proxies for the mobile applications. In each case, the developer is asked for essential parameter names through a dialog box and relevant code is inserted.

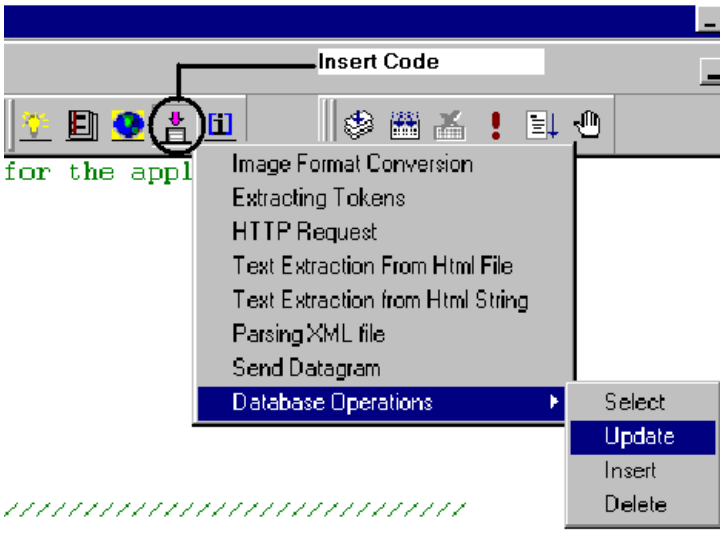

Figure 3 - MobileToolbar Code Generator Button

Following options are provided in this case:

- Image Format Conversion - It inserts code for converting image from one format to PNG format and asks for source and destination filenames.
- Extracting Tokens - It inserts code for extracting tokens out of a string based upon a delimiter. It only asks developer for the object name of StrManip class, source which needs to be tokenized and delimiter on whose basis source needs to be tokenized.
- Http Request - It gives developer dialog box to enter object name for class httpQuery, source from where to fetch the data, filename in which to fetch the data and the interval after which the data needs to be fetched.
- Text Extraction from the Html File - This option asks developer for object name of HtmlTOText class, filename from where data needs to be extracted, tag name and, start and end position for the data to be extracted from string.
- Text Extraction from Html String - This is similar to the previous option; the only difference is now source is a string instead of a file.
- Parsing XML File - In this case, developer is asked about the XML file, which needs to be parsed, the element name and the attribute name whose value is required. Following comments are inserted in the code which tell the developer which variables contain the element value and the attribute value -

//Use bstrItemText which contains the Element text

//Use bstrItemText which contains the Attribute Value

- Send Datagram - Dialog box will ask the developer for object name of the DtgrmSendRcv class, string to be sent and the destination IP address and the port number.
- Database Operations - The developer is provided with four options - Select, Update, Insert and Delete - pertaining to data manipulation as well as data retrieval of the database. In each case, the developer is asked for CDatabase class object and the corresponding statement.

In all of the above cases, code inserted is also formatted and takes care of where it is getting inserted. With the Insert Code feature, the developer needs to know very little about the API we have provided and can still make use of it.

5. About Toolkit - This is the last button, which gives information about the whole toolkit.

3.4. MOBILEPROXY

MobileProxy is a custom Appwizard that creates a proxy shell, depending upon the options selected by the developer. This wizard appears where the rest of the standard Visual C++ IDE 6.0 wizards are included and can be invoked by clicking File > New > MobileProxy Appwizard. As the developer selects *MobileProxy* AppWizard and gives a project name, wizard provides different options to the developer. Figure 4 shows first list of options provided to the developer in first step.

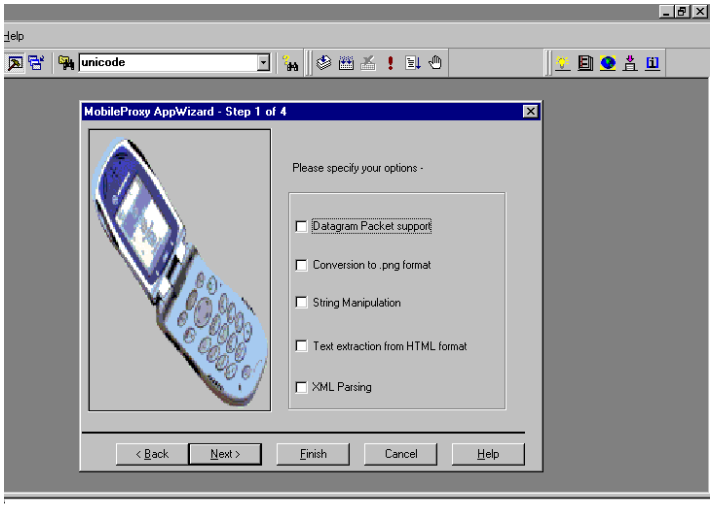

Figure 4 – API Options in Step 1 of MobileProxy

Following options are included in MobileProxy through its four steps.

1. Datagram Packet
2. Conversion to PNG format
3. String manipulation
4. Text Extraction from HTML format
5. XML parsing
6. HTTP access to a website
7. Web Services Access
8. Database Support
9. User Profile Management
10. Address Management
11. Client Property Management

For all the above options, except for the Database Support option, appropriate headers files are included in the generated proxy shell files. Figure 5 shows the Database support dialog box.

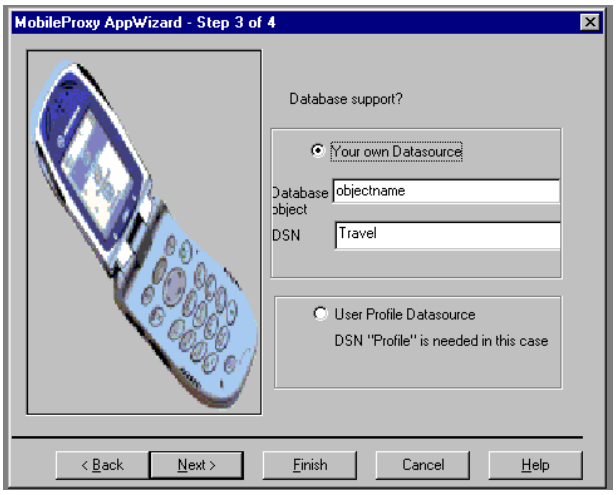

Figure 5 – Database Options for developer in Step 3 of MobileProxy

Last option for the developer to choose from is the target client device whose display size and display color is to be included in the project. The wizard generates the following files in the project:

- Listener.cpp and Listener.h - This provides all the functionality of server, which continuously listens for the client requests at a particular port. It makes use of datagram socket for this purpose. Whenever it receives a datagram packet, it creates a new thread that in turn creates a new Worker class object and passes it the data along with the client's IP address and port number.
- Worker.cpp and Worker.h - This class has run() method which is called by the thread initiated in Listener class, whenever the proxy receives datagram from the client. So, run() method of this class should include logic for processing the client request. It is provided with the data sent by them client as well as the client's IP address and port number. Worker.cpp includes constants DISPLAY_HEIGHT and DISPLAY_WIDTH, which contain the value of the target machine's display height and display width. It also contains constant DISPLAY_COLOR, which states whether the target device's display can support colors or not.
- dbHandler.cpp and dbHandler.h - dbHandler file is included only if the developer makes use of the option Database Support and wants to connect with a DSN(Data Source Name) using ODBC. This class creates a CDatabase object whose name is provided by the developer and also provides all the logic to connect to DSN using ODBC. This class should contain all the methods to manipulate the database.

4. EMPIRICAL ANALYSIS

The goal of this analysis is to find how much benefit the *ProxyGator* toolkit has provided for writing a typical mobile application. To find that, we have counted the total number of lines of code in the proxies of each application from where we derived our API requirements. Then, we have counted the approximate number of lines of code that can be written by the API provided or can be addressed by MobileToolbar and MobileProxy.

4.1. Analysis

Table 1 shows lines of code that can be addressed by API, MobileProxy and MobileToolbar together.

Table 1 – Percentage of code addressed by the ProxyGator toolkit

Application	Total lines in Proxy	Lines by Toolkit(approx)	Percentage
AirTicket	485	126	26%
MapIt	2090	943	45%
WeatherClient	3049	1212	40%
StockDen	630	301	48%
SunShine	715	358	50%
SmartShop	1192	336	28%
TravelExpense	1570	591	38%
WeatherView	787	104	13%
iCoffee	1478	86	6%
CellMoney	211	10	5%

From the table, we have made few observations:

- In 7 out of 10 applications, this toolkit addresses 25% or more of code. Since, the API methods replace multiple lines of code in almost all the cases, usage of the API would considerably reduce development time and will also make the developer's code smaller in size. Additionally, it will reduce a lot of debugging effort. Through the *API Help* available in *MobileToolbar*, a developer can easily find the required method. Further, in many cases, code can be inserted using *Insert Code* feature of *MobileToolbar*.
- For those applications that make use of database, the toolkit without database support wouldn't have been of much use. In these applications, the database support constitutes up to 50% of total support.
- For remaining 3 applications, the API provides very little support - 13% or less. The reason for this is that these applications are not web intensive applications. iCoffee project is making use of specific hardware and its proxy code is written to drive it. CellMoney project makes use of Java Servlets, although, the same functionality can be provided by the use of this API.

5. CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

In Section 4, we concluded that the *ProxyGator* addresses up to 50% of lines of code in a typical mobile application. As proxy is an indispensable part of most of the J2ME based mobile applications, the usage of this toolkit would greatly improve the development time of a typical mobile application.

As the toolkit is developed in VC++ and integrated within the Visual Studio IDE, the usage of toolkit increases further because of the large developer base of Visual Studio as well as a range of features Visual Studio provides to the developers. One key feature of the toolkit is its support for Web Services. Although still in its nascent stage, Web Services are going to take development of applications to a new level, whereby programs search for other programs and create an application. As this support is included in the toolkit, it will increase its usage and also empowers the developer.

One of the limitations of this toolkit is that the client of the toolkit should be an MFC application. Although, MFC provides a range of features, its learning curve is higher than Java language. Secondly, a

developer needs to learn what is present in the API and how to use it along with *MobileToolbar* and *MobileProxy*. These limitations are addressed by providing proper help through *API Help* feature of *MobileToolbar*, which details all the classes and their methods present in the API.

5.2. Future Work

This is just a beginning of infrastructure provision for the development of proxies for J2ME platform. The range of support that can be included in this toolkit is just limitless. We now enlist some of the support this toolkit may include in the future:

1. Secure Socket Layer Communication - Sometimes the data coming from the client to the proxy and vice versa needs to be on the secure channel. Similarly, when the proxy communicates with the server, it should be secure. This is done when usernames or passwords or credit card numbers or any other personal information is sent on the channel. As SSL is being introduced in J2ME based mobile devices, future toolkits can contain an easy-to-use API for such types of communication. Having secure communication between the proxy and client or server is very important for serious business applications.

2. Link Quality Measurement - Quality of wireless links doesn't remain the same and depends upon movement as well as position of the user. An API which can measure link quality between the proxy and the client would really help in designing applications which adapt according to the changing bandwidth, maximum transmission unit (MTU) of datagram packets, jitter and latency of the link. This way the proxy can limit the amount of data to be sent to the client if it comes to know that link quality is poor. This will help in giving quicker response and avoid wastage of resources.

3. Measuring client capabilities - Although we are providing option to choose between different handheld devices while the creating proxy, measuring device capabilities on the run will help in designing applications that can target multiple devices and transform data according to the capabilities of the client. Some of the capabilities that can be measured are - display area, graphics capabilities and processor speed.

4. Support for TCP - Right now we are only providing support for UDP communication between client and proxy through *MobileProxy*. But providing a choice between TCP and UDP for communicating between client and proxy will address those applications, which communicate using TCP.

We hope that the ProxyGator toolkit will help in quick development of the mobile applications and address the needs of the developers.

6. REFERENCES

[1] Margaritis Margaritidis and George C. Polyzos. Adaptation techniques for ubiquitous internet multimedia. Journal on Wirelss Communications and Mobile Computing, 1(2):141-163, January 2001.

[2] M. Naghshineh and Marc Willebeek-LeMair. End-to-end QoS provisioning in multimedia wireless/mobile networks using adaptive framework. IEEE Communications Magazine, 35(11):72-81, November 1997.

[3] Mobile computing applications. http://www.harris.cise.ufl.edu/killerapps/S01/demos.html, May 2001.

[4] 2PNG software, http://batch.fcodersoft.com/2png/.

[5] Roger Wolter. XML Web Services Basics. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/webservbasics.asp , December 2001.

[6] Ariba Inc., International Business Machines Corporation, and Microsoft Corporation. UDDI technical white paper. http://www.uddi.org/whitepapers.html, September 2000.

[7] Ariba Inc., International Business Machines Corporation, and Microsoft Corporation. UDDI Programmer's API Specification. http://www.uddi.org/pubs, September 2000

[8] Francisco Curbera, David Ehnebuske, and Dan Rogers. Using WSDL in a UDDI Registry 1.05. http://www.uddi.org/ , June 2001.