Protocols for Mobile Devices Integration in Heterogeneous Environments

Anatoliy Doroshenko [1], Kyrylo Yatsenko [2]

[1] Institute of Software Systems of NASU and National University of "Kiev Polytechnic Institute",
Glushkov prosp 40, block 5,
03187 Kiev, Ukraine,
dor@isofts.kiev.ua

[2] Cybernetics faculty at the University of Taras Shevchenko
Glushkov prosp 2,
author@kirillyatsenko.com

Abstract. An approach to construction of new class of cross-platform communication protocols XDEP (XML Data Exchange Protocol) is proposed. The goal is to facilitate specification-independent development of the client software for mobile devices in heterogeneous environments. The new class of the protocols is self describable which does not require providing specification documents to software developers. This makes also possible to provide multiversioning of protocols to be used on the same devices.

1 Introduction

The world of mobile devices started its incredible growth with invention of Palms and BlackBerries. "Pilot" was the name of the first generation of personal digital assistants manufactured by Palm Computing in 1996 (http://en.wikipedia.org/wiki/Palm_%28PDA%29). In the very begging of their history the mobile devices had very limited amount of memory and processor's frequency. Nevertheless, some time later this market has experienced revolution that received new players like Microsoft, Hewlett Packard, Dell, and others. PocketPC has become the successor of Palm's technology which has got additional resources and "know-how" from big corporations. Though the first devices were considered only for personal or private use, shortly wireless technologies developed in parallel opened a new market for these devices – corporate solutions. New software has started integration of mobile systems in its business processes: emails, integrated GSM/CDMA protocols and other advanced features allowed creation of mobile offices right in "a pocket". This breakthrough has been considered with great enthusiasm until the software developers met serious difficulties during the integration of heterogeneous systems. Integrated cell phone protocols has provided additional methods of communication, but communication

establishment between device has made more difficulties considering that these more advanced features were present not in all devices.

In parallel to the hardware development software development has made its own breakthrough with various communication technologies like CORBA, COM, CGI, web-services and many others that promised reductions of developers' efforts during the construction of distributed system. However, only few of them gained their popularity in desktop applications, and to the end *only* web-services migrated to mobile devices. The most significant here is that COM technology, once announced as the final point of integration of all Microsoft products, was not transferred even to contemporary Windows Mobile 5.0 system (which was the third version of PocketPC platform since the first one published in 2002). However, web-services require "heavy" third party software installed on the client devices that made it unacceptable because the libraries like Microsoft .NET Compact Framework could take till 80% of the default device's memory. Thus developers where obliged to implement communication between mobile and server application using simple sockets or ATP (GSM) modems.

It is worth to mention that web-services are built on text protocols. With invention of XML text format has become popular again. All the data exchange technologies other then web-services use binary format that causes difficulties in development and system migration. As to the text format, the greatest disadvantage is a big volume of the technical data. However, benefits that it brings cannot be overestimated. The main advantage is simplicity of the development and usage. Text formats became popular not only in the computer networks but also in protocol of communication between different devices, for example the protocol of NMEA (National Marine Electronics Association) (http://en.wikipedia.org/wiki/NMEA).

1.1. Review of existing data exchange solutions

One of the main goals of heterogeneous system development is making decision on communication methods between devices with different logical and physical structure and their integration into a single system. The difficulties related to this integration come from differencies in software and hardware environments. The network sockets are the most commonly used solution for these issues. Being a universal approach this solution is not efficient in every particular case.

Common Gateway Interface (CGI) is the protocol that identifies the process of HTTP-server and routing software interaction [G96]. In other words, protocol CGI – is the gateway between HTTP-server and software that cannot be directly called via HTTP protocol. CGI software is run by web-server that provides hosting of a system and transfers requests received from a client application to CGI applications that process these requests and return processing results to the client via web-server. CGI – application can be an interpreted script (like PHP, Perl, etc.) or a compiled program (written with C/C++, Visual Basic, etc). This technology has few key features. The first one is that it does not require any special software installed on the client machine as CGI-applications are totally executed on the server side and do not depend on client computer and each other. Here arises the second peculiarity which is the absence of any

supervision methods over CGI-applications that share common resources. Therefore it is difficult to use this technology for construction of complex systems.

Another data access technology is an Application Programming Interface (API) that is a set of rules that identify procedure calls with the help of program packages [S00]. This is a standard technology for local computer programs, but as an Internet-technology it has started to be used only recently. For example, in the COM (Component Object Model) API is compound of a single or few dynamic libraries DLL (Dynamic Linked Library) and a COM component that provides an access interface to them. By contrary to CGI, API assumes package integration with the whole system. This technology is much more complex then CGI, but at the same time it is more powerful one. If CGI provides a gateway with which client application may get access to different processes running on the server, API provides methods for remote applications to run program functions directly without any mediators.

Here, there are few important moments. The first, the API technology requires special software installed on the client computer. The second, it provides special mechanism to control all system components. In addition, this system should be integrated with operating system and use some of its program components which improves efficiency of such programs. [R97] It is possible to consider that CGI technology allows construction of multiple stand-alone applications that are working independently one from each other, and the API, conversely, integrates a system which consists of different components. Comparing to API, CGI programs work slowly and require more server resources. However they are not demanding to client software as API is, which requires special installations on the client machines. [T02]

With appearing of distributed computation, network interaction has reached the next evolutional stage where different companies could coordinate their efforts and more reasonably use their resources by using possibilities of each other. That way appeared to be web-services that have become the most advanced data exchange gateway. The W3C defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network. Web services are frequently just application programming interfaces (API) that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. The W3C Web service definition encompasses many different systems, but in common usage the term refers to those services that use SOAP-formatted XML envelopes and have their interfaces described by WSDL. For example, WS-I (http://en.wikipedia.org/wiki/WS-I) only recognizes Web services in the context of these specifications.

Security mechanisms of SOAP protocol are worth of special attention. In case of small projects security analyzers supported in SOAP by default would be enough. But for the development of robust and large Internet projects with distributed objects, schemas, and access scenarios, developers will have to tackle a problem of a poor set of default security features. Thus, they will be obliged to develop their own security features like datagram analyzers, filters, etc. Another disadvantage of SOAP is low operation speed, which is more complex problem, as developers can not change protocol specification in order to reduce response delay. This intrusion would impact communication methods

and as the result would disable usage of devices that are missing the changes in the protocol implemented on the server. So the sense of web-services would be wasted. Even, if the developers decide to increase bandwidth of the Internet channel every low bandwidth server outside of the company would impact the whole system, becoming a bottleneck of the solution. And the last thing about web-services is their ambiguous specification. Different environments may interpret system call differently causing necessity of additional system verification and testing. Thus, web-services developed in one organization may be incompatible with another web-service outside of it.

1.2 An alternative to CGI, API, and web-services

Products supplied by Microsoft [Fog03] and Sun Microsystems [Gra04] have become standards *de-facto* for development of web-services.. Both platforms have very common development approaches. The main requirement for both systems is presence of special software installed on client application machines. Integration of web-services with portable devices raises some difficulties because of their capacity and performance. Certainly, third-party libraries installation can be possible, but by default even Microsoft PocketPC and Microsoft Windows Mobile operating system do not contain MS .NET Compact Framework, which is required to access to web-services. When an enterprise wants to develop its own hardware numerous issues rise with implementation of web-service clients.

This paper proposes an alternative to web-services light-weighted approach to construction of a new class communication protocols. Comparing to web-services the new class is more persistent and more flexible in involving of wide range of devices. Also, it can be easily adapted to the specific communication methods of devices and applications. The customer can exploit third-party XML libraries or instead their XML-like software to support text mode of communications. In the second case the protocol promotes compact implementation of communication methods because it does not require installation of web-service third party libraries. The new class of the protocols is self describable which does not require providing specification documents to software developers. This makes possible to provide multiversioning of protocols to be used on the same devices.

Provided examples and solutions are based on the Microsoft® technologies; however the class of the protocols developed is not limited to this platform. It can be successfully applied to computers with various architectures.

2 XDEP. A class of protocols

XDEP (XML Data Exchange Protocol) is a class of application level protocols for data exchange that implements authentication and authorization. Comparing to web-services specification, which is compound of numerous modules:

- SOAP (http://en.wikipedia.org/wiki/SOAP): an XML-based, extensible message envelope format, with "bindings" to underlying protocols (e.g., HTTP, SMTP and XMPP).

- WSDL (http://en.wikipedia.org/wiki/WSDL): an XML format that allows service interfaces to be described, along with the details of their bindings to specific protocols. Typically used to generate server and client code, and for configuration.

- UDDI (http://en.wikipedia.org/wiki/UDDI): a protocol for publishing and discovering metadata about Web services, to enable applications to find Web services, either at design time or runtime.

- WS-Security (http://en.wikipedia.org/wiki/WS-Security): defines how to use XML Encryption and XML Signature in SOAP to secure message exchanges.

- WS-ReliableMessaging (http://en.wikipedia.org/wiki/WS-ReliableMessaging): a protocol for reliable messaging between two Web services.

- WS-Reliability(http://en.wikipedia.org/wiki/WS-Reliability): an OASIS standard protocol for reliable messaging between two Web services.

XDEP can be described only with few rules, which are sufficient for the development of applications with reach communication abilities.

The main feature of XDEP is implemented session support that doesn't require constant real-time connection. The class also describes logic requirements to communication methods and data structures in heterogeneous environments. Tasks that can be successfully resolved during usage of the protocol are

- Multiplatform client applications;

- Data safety from unauthorized access.

Similar to web services [Fog03] XDEP constitute the method of integration based on a few layers. The first layer is the DAL (Data Access Layer) where programs exchange data only. For instance, a program running on one server may process data located on another one. This is the simplest integration level. Further is the object interaction layer. On this layer applications talk the same "language" bypassing architectural differences. The second layer of integration requires only semantics' standardization. This standardization is achieved via using XML syntax, common within the applications. Client and server applications just supply data, and adopted for different platforms XDEP libraries implement the data exchange.

In addition the class of the protocols is compounded of global and session access levels. Each access level defines type of data provided to the client application. For instance, technical information regarding server's hardware can be obtained without client authentication. This type of information belongs to global access level. In the meantime, data from the database can be interpreted as sensitive data, thus only authorized

application can be granted access to it. Server generates responds to every request received from the client. Independently from the request's processing results, server has to confirm request processing. Below there presented the schema of requests and responses for global and session level datagram.

2.1 Global access level

Commands of this type provide access insensitive data, information accessible by any software that is able to compose XML documents (request to the system).

Command's schema. Format of the commands of global level are:

```
<[CommandName] Version="string" RequestID="string" devid="deviceid">
        <[RequestSpecificParams]>
        </[RequestSpecificParams]>
</[CommandName] >
```

[CommandName] – is the name of the command that is unique in scope of the logical system. Every protocol of XDEP class supports command versioning, which allows commands handling with the same name but different schemas, in other words provides mechanism of the commands polymorphism. For request identification in the framework of the server every command must be supplied with the version number. This information is committed through the Version tag of the CommandName attribute.

[RequestSpecificParams] is the XML content of a request. The specification on XML doesn't limit the hierarchy of the attributes, thus the command doesn't limit the structure of the information transferred via XDEP. The attributes of the command are considered in the Table 1.

Name	Short description	Default value	Optional
Version	The version of the command	Current version	Yes
RequestID	Unique request identifier in the client application	None	No
DeviceID	Unique client identifier in scope of the server application	0	Yes

Table 1. Command Attributes

RequestID is the unique identifier of every command submitted to the server application. This attribute is required only for client application, as in case of parallel transactions, the software have to distinguish different commands. Regarding the server application, this attribute is ignored because the server has to have internal command identification and processing mechanism.

DeviceID is the unique identifier of every client device. Certainly, the server is responsible to distinguish client applications for the security reasons. This value should be hard-coded into every client application, thus to make it static. The server in responsible to track these identifiers and generate them in any format that is convenient to it.

Responses schema.

<[CommandName]Resp RequestID="string“ Code="000" Msg="OK”>
    <[RespSpecificParams]>
    </[RespSpecificParams]>
<[System]>
    <[SystemSpecificParams]>
    </[SystemSpecificParams]>
<[/System]>

[CommandName]Resp, equally to the name of the command, is the tag that represents the name of the response, which is combined of the CommandName and Resp suffix.

[RespSpecificParams] is the body of the response.

[System] − is worth of special attention in this schema. The disadvantage of the most of communication protocols is absence of direct server to client calls. This issue arises because of the peculiarity of sockets implementation where one part must be a server and another is client application. Another reason of inability to establish connection between the server and the client (initiated by the server) is firewalls and proxy server used everywhere today. Thus, the only way to transfer information by the server's request is the case of the response. The tag *System* has been added especially for this purpose. If server needs to "say" something to the client software, all these data has to be put under the *System* attribute. For instance, if the server is intended to move to another IP address or domain name, all client application has to be informed about this event. Table 2 contains description of the attributed and parameters used in the Response Schema

Name	**Short description**	**Default value**	**Optional**
RequestID	Request unique identifier in the client application	0	No
Code	Error code	0	No
MSG	User friendly error description	Empty string	Yes

Table 2. Response attributes

Differently to the request, the response command contains specific attributes that can should be analyzed by the client application in order to identify whether the request has been processed successfully.

[Code] – is the error code raised on the server. This value is used by the client software for the error identification. Below in the table 3 there is a list of the standard errors, that should not changed during the proper protocol implementation.

[MSG] – is the error description in the user friendly format.

2.2 Session level commands

Commands of this level provide access to the client application to the sensitive data. The user is not obliged to establish on-line connection with the server in order to have open session with the server. The server is responsible to provide session identifier to the client application that will be expired in some time. The new session starts with an *OpenSession* request send by the client application. Responding to this message, the server generates the session identifier, which provides to the user. Important note here is that the server verifies device ID and if it is valid then handles the command.

```
<OpenSession Version=”string” RequestID="string"”>
        <devid></ devid>
        <username></username>
        <password></password> [encoded]
</OpenSession >
```

<devid> node contains unique system identifier.
Response to the *OpenSession* request is below:

```
<OpenSessionResp RequestID="string" Code="000" Msg="OK">
        <SessionGUID>123kjhsad908h@#q389qgm
        </SessionGUID>
</OpenSessionResp >
```

The server is responsible for session identifier generation. On the level of the protocol, this value represents a simple text string, thus it can consist of any value that is convenient for the server. Using this session identifier the user will be able to post session level requests.

Session level request

```
<[CommandName] Version="1" RequestID="31“ SessionGuid =“string” devid=”device id”>
        <[RequestSpecificParams]>
        </[RequestSpecificParams]>
</[CommandName] >
```

Name	Short description	Default value	Optional
Version	Request version	Current version	No
RequestID	Request unique identifier in the client application	0	No
SessionGuid	Session unique identifier	None	No
deviceid	Unique client identifier in scope of the server application	None	No

Table 3. Session level request attributes

Response schema of the session level requests is equal to the schema of the global level request.

2.3 Error Handling

XDEP protocol standardizes error handling and provides the list of the standard errors. The codes of the standard errors are between 0000 and 0999. Table 4 lists some of these errors.

Code	Short Description
0000	No error
0001	Unknown request
0002	Attributes format error
0003	Attribute parameter error
0004	Nodes format error
0005	Absent close nodes
0800	Unable to open session
0801	User or password is invalid
0802	Unknown session identifier
0998	Server error
0999	Unknown error

Table 4. Some of the standard errors

2.4 Data encryption

The protocol implements special tags for the for the data encryption. <Encrypted> node hides the data encoded in the system's format. For every protocol the encryption system differs depending on the implementation. Node <Encrypted> may be added to the both level of the commands. The value of the node is the encrypted data.

<Encrypted>
[< Content >]
[</Content >]
</Encrypted>

2.5 An Example of Using XDEP

To make it clear usage of the protocol for production purposes as an example is considered NMEA protocol, that has been mentioned in the begging of the article. The message below is a sample message extracted from the dialog between smart device and a Bluetooth GPS receiver:

$GPRMC,040302.663,A,3939.7,N,10506.6,W,0.27,358.86,200804,,*1A

There are few disadvantages of this protocol. In order to verify that the message has not been corrupted it is required to calculate a Checksum (http://en.wikipedia.org/wiki/Checksum). This additional verification is needed because the data is formatted into a single string without strict structure. The checksum is performed via comparing the length of the string with the value after the "*" symbol in the hexadecimal format. Another disadvantage of this format is a human - unfriendly layout.

Both these issues is possible to resolve by making the protocol self describable. The sample message can be translated to another format based on XDEP rules.

<twreq stamp="rmc" devid="device id"[queryresult=1]>
<lat d="N | S">latitude</lat>
<long d="E | W">longtitude</long>
<sats>A | V</sats>
<satt>satellite timestamp</satt>
<clt>client side timestamp</clt>
<spd>horizontal speed </spd>
<dirc>true direction of mobile client </dirc>
[<alt>altitude</alt>]
</twreq>

Where "Stamp" means the GPS message stamp (http://nacs.de/schiffel/nmea0183/rmc.html), "deviceid" is the device's identifier added according to the XDEP protocols specification.

3 Conclusion

This paper has introduced a new class of self-describable protocols XDEP for data exchange based on the XML format as an alternative to web-services. The class intends to provide standards of user's own protocols development for data exchange in heterogeneous environments. Comparing to web-services the new class is more

persistent and more flexible in involving of wide range of devices. Also, it can be easily adapted to the specific communication methods of devices and applications. The feature of self describability of the protocols does not require providing specification documents to software developers. This makes possible to provide multiversioning of protocols to be used on the same devices. The approach of XDEP is not universal like web-services, but provides some rules for development of user's own protocols, which is more flexible, though, requires additional development efforts. These protocols can be better integrated into a single system more smoothly comparing to web-services. This feature makes XDEP interesting solution for mobile system.

The first commercial application of XDEP has been related to construction of the system with numerous mobile PocketPC devices with embeded GPS receivers. This experience has shown pros for XDEP mentioned above along with its cons (about 60% of every request is a technical information), which are seemed to be more and more insignificant in perspective.

Bibliography

[Gra04] Graham, S; Davis, D.; Simeonov, S.; Daniels, G.. Building Web Services with Java : Making Sense of XML, SOAP, WSDL, and UDDI (2nd Edition) (Developer's Library), 2004

[Gu96] Gundavaram, S.: CGI Programming on the World Wide Web (Nutshell Handbook), 1996

[Fog03] Foggon ,D; Maharry D.; Ullman, C.; Watson, K.; Programming Microsoft .NET XML Web Services (Pro-Developer), 2003.

[Ro97] Rogerson, D.: Inside Com (Microsoft Programming Series), 1997.

[Sw00] Swanke, J: COM Programming by Example: Using MFC, ActiveX, ATL, ADO, and COM+ (with CD-ROM), 2000.

[Tr02] Troelsen, A..: COM and .NET Interoperability, 2002.