Web Services in Mobile Applications

Octavian DOSPINESCU[1], Marian PERCA[2]
[1]Al.I.Cuza University of Iasi, Romania
[2] The Software Farm, UK
doctav@uaic.ro, marian.perca@gmail.com

Information and communication technologies are designed to support and anticipate the continuing changes of the information society, while outlining new economic, social and cultural dimensions. We see the growth of new business models whose aim is to remove traditional barriers and improve the value of goods and services. Information is a strategic resource and its manipulation raises new problems for all entities involved in the process. Information and communication technologies should be a stable support in managing the flow of data and support the integrity, confidentiality and availability. Concepts such as eBusiness, eCommerce, Software as a Service, Cloud Computing and Social Media are based on web technologies consisting of complex languages, protocols and standards, built around client-server architecture. One of the most used technologies in mobile applications are the Web Services defined as an application model supported by any operating system able to provide certain functionalities using Internet technologies to promote interoperability between various applications and platforms. Web services use HTTP, XML, SSL, SMTP and SOAP, because their stability has proven over the years. Their functionalities are highly variable, with Web services applications exchange type, weather, arithmetic or authentication services. In this article we will talk about SOAP and REST architectures for web services in mobile applications and we will also provide some practical examples based on Android platform.
Keywords: *Mobile Web Services, Mobile Applications, SOA, SOAP, REST*

1 Introduction

The distributed computing systems become more and more important in the new world dominated by the Information Technologies. This has resulted in recent standardization effort of distributed computing architecture, which is known as Service Oriented Architecture (SOA). The main component of this architecture is the Web Service. Some of the challenges in implementing the SOA architecture are maintainability, reliability, and security [1].

In the specialized literature we can find several articles [2], [3], [4] that address the issue of distributed architectures and web services in areas such as education, health and business. In fact, Web services are a solution for the integration of distributed information systems, autonomous, heterogeneous and self-adaptable to the context. Although there are similarities between Web Services and Web sites, we mention three notable differences between these technologies:

- websites have generally a friendly interface to communicate with the user and the opposite we have web services that do not require a graphical user interface;
- if Web sites are designed to facilitate interaction with their users, web services will only exchange data with other applications;
- websites must be accessible using browsers while web services can adapt to several environments or devices.

An application built on web services architecture can have several roles; it can be both the consumer and the service provider or just register whose role is to keep the web service description. The Service Provider is the person or organization that provides access to a web service in order to meet certain requirements. The consumer (Service Requester) is the entity that needs this service and meets a register known as the Discovery Service Agency or Broker who will make the connection between the two components [5].

In this paper we intend to present a review of the main architectures based on web services

DOI: 10.12948/issn14531305/17.2.2013.02

and we also provide an application implementation which can be used in the industry of telecommunication.

2 The Components of the Web Services Eco-System

According to some authors [6], web services were initially mainly intended to engage in dynamic business-to-business (B2B) interactions with services deployed on behalf of other enterprises or business entities. Broad interest in standardization/customization efforts was aimed at reducing the necessary user interaction. However, with the advancement of Web service technology the complexity of possible tasks and the availability of services any time anywhere, e.g. through powerful mobile client devices, will strongly increase.

The architecture of calling services is based on its description as shown in Figure 1.

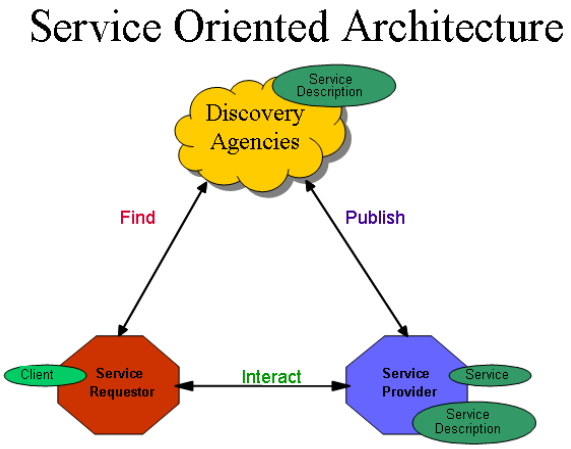

Fig. 1. The architecture of calling services [7]

The main building blocks of a web service are discussed below, along with a description of each and the function that satisfies thus making up web service stack:

- **Discovery** - the process that determines the location of the Web service that will connect. The discovery will be done using centralized directories or using other ad-hoc methods;
- **Description** - web service discovery once made, the customer must receive a description of how it will achieve interaction with the service. Description consists of structured metadata about the interface used and written documentation of details and examples of using the web service;
- **The format of the messages** - this element is necessary for encoding the message between the client and web service. This format allows abstraction of communication protocols for a better focus on the logic of the problem;
- **Encoding** must allow processing by any language; XML is the best choice due to the structural and text;
- **Transport** - how data composing a message is transmitted between partners in conversation.

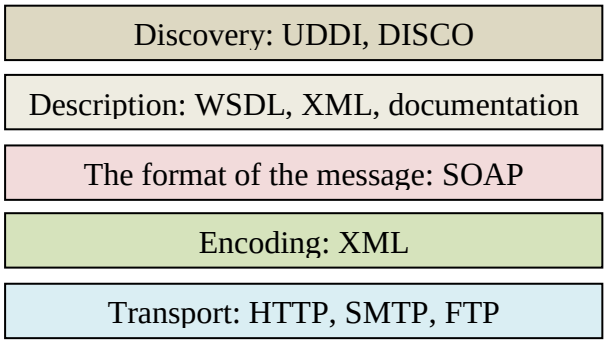

Fig. 2. The main building blocks of a web service

Some authors [8], [9] consider that SOAP based Web services are the preferred way to implement the SOA initiative in today's complex and heterogeneous computing environment. SOAP-based Web services present greater flexibility at lower integration costs over RESTful Web services that instead offer great performance through lighter messaging system.

Instead, we consider that the applications that use REST web services are more robust and the costs are not a problem for the implementation.

3 Types of Architectures

Although there are numerous standard protocols such as HTTP, XML, WS-Notifications, WS-Security, Simple Object Access Protocol and Web Service Definition Language to simplify Web services, their definition was chosen based on the type of service. The

DOI: 10.12948/issn14531305/17.2.2013.02

RPC Architecture (Remote Procedure Call) is a service-oriented and developed around the idea of trading information via a wrapper type SOAP, XML-RPC or HTTP. An alternative appeared as REST architecture resource-oriented which exclusively uses web technologies such as HTTP, URI or XML.

3.1 RPC Architecture – Service Oriented Architecture Protocol

Remote Procedure Call is an inter-process communication technology that allows a program on one computer to generate a subroutine or procedure which is executed in another address space (usually on another computer or on a shared network) without the programmer explicitly encode the details of these interactions at a distance. Basically, the programmer writes the same code whether the subroutine is local or remote to the executing program. When the software in question is written using object-oriented principles, we can talk about speed or distance calling remote methods.

One way to examine the architecture of web services is achieved by studying the protocol stack of the service. Stack is growing but currently contains four levels:

- **the transport service**: the message is responsible for transportation between consumer and supplier. Currently this level contains protocols such as HTTP (hypertext transfer protocol), Simple Mail Transfer Protocol (SMTP), file transfer protocol (FTP), and newer protocols like Blocks Extensible Exchange Protocol (BEEP);
- **the level of XML messages**: this is responsible for encoding messages in an XML format so that the messages can be understood from the other end. Currently this level is described by XML-RPC and SOAP protocols;
- **the level of service description**: is responsible for describing the public interface of the service. Service description is achieved by Web Service Description Language (WSDL);
- **the level of service discovery**: this is responsible for centralizing services into a common registry and provides publishing functionality of the service. The discovery service is currently done by: Universal Description, Discovery, and Integration (UDDI).

SOAP is an XML-based protocol that exchanges information between computers in a decentralized and distributed environment. The acronym SOAP originally derived from Simple Object Access Protocol, and then from Service Oriented Architecture Protocol. The initial name was abandoned from version 1.2 when it was considered misleading. The protocol consists of three elements:

- an envelope that defines the content of a message and how to process it;
- a set of rules for encoding instances defined by the application (defined data types);
- a convention for representing procedure calls and responses called away.

Like XML-RPC, SOAP is platform independent, thus representing a way of communication between different operating systems with different operating languages. SOAP messages are encoded in XML documents, are made up of a wound (SOAP envelope, mandatory), a header (SOAP header, optional) and a body (SOAP body, mandatory). SOAP messages are also XML documents, giving details of the protocol specifications for encoding data in strongly typed SOAP messages. The figure below summarizes the structure of a SOAP message:

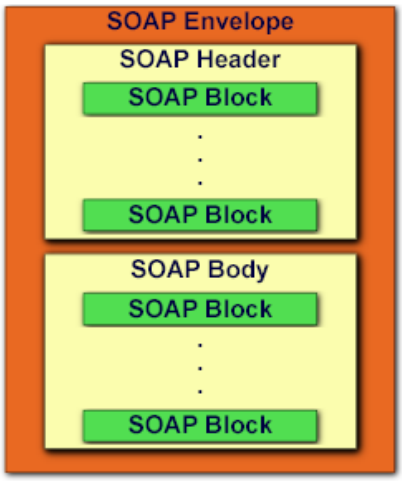

Fig. 3. The structure of a SOAP message[10]

DOI: 10.12948/issn14531305/17.2.2013.02

3.2 REST Architecture

Representational State Transfer (REST) is a new architecture for web services that is having a significant impact on the industry. Most of the new public web services from large vendors (Google, Yahoo, Amazon, Microsoft) rely on REST as the technology for sharing and merging information from multiple sources [11]. Representational State Transfer involves a set of rules or a resource-oriented architecture that supports the simplicity of web technologies, using standards such as Hypertext Transfer Protocol Uniform Resource Identifier and Extensible Markup Language. Among the most notable features of the REST architecture, we can mention [12]:

- the data transmitted from the client to the server is in URI;
- the operation performed by the server on the data is described in the HTTP method directly;
- resource - anything that can be labeled as object (concrete or abstract) will automatically be a resource;
- URI for each resource that contains her name and address. In most cases this URI is identical to the Uniform Resource Locator (a basic form of identifier);
- representations - their core resources will not represent data, but only the idea on the structure of the data service.

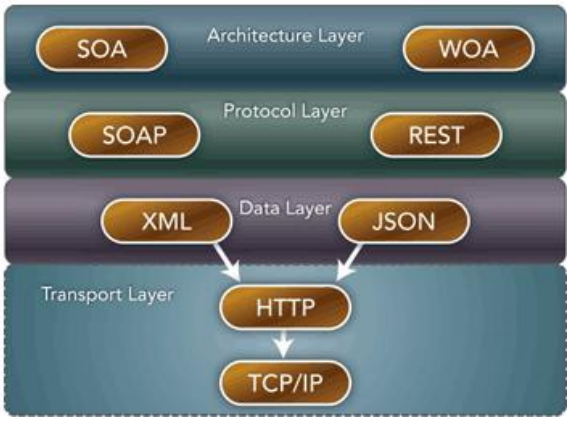

Fig. 3. The layers of SOAP and REST [11]

According to some authors [13], the HTTP methods such as GET and POST are the verbs that the developer can use to describe the necessary create, read, update, and delete (CRUD) actions to be performed. Some may see an analogy to operations in SQL, which also relies on a few common verbs (Insert-PUT, Select-GET, Update-POST, Delete-DELETE). However, the REST style and HTTP protocol are mutually exclusive, and REST does not require HTTP.

4 A Model of Mobile Application Using Web REST Services

In this section we implement an Android mobile client which uses REST services in order to obtain information about the phone calls for a company that has many mobile devices. The mobile application must meet economical ([14], [15]) and technical requirements:

- covers many mobile device types, running at least Android 2.3.3;
- authentication of the same screen of companies, divisions and simple users;
- access to available reports for each user, depending on the selected reporting period;
- viewing data as graphs;
- viewing messages.

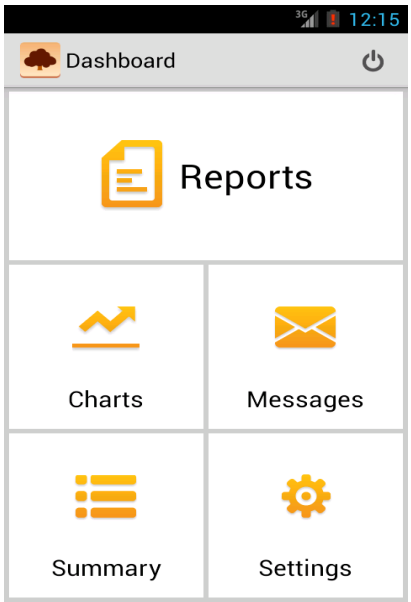

Fig. 4. The sections of the application

To perform these functions, we divide the application into 5 sections:

- Reports - reports can be viewed here;
- Charts - to view graphs;
- Messages - for viewing messages;

DOI: 10.12948/issn14531305/17.2.2013.02

- Summary - to view data on current account;
- Settings - to set the year and used in the reporting month.

The data presented by the mobile application is through the REST web services offered by the web application. We used the technique described in [16] and we add some new features in order to meet the requirements described in [17]; this involves calling a web service through an Android service that does not block the user interface. There are many advantages of using a service, among which the most important are the following:

- the service does not depend on a specific activity, so we can start the services from different parts of an android application, including other services or BroadcastReceivers;
- the service does not block the user interface, making it ideal for operations that take place on a longer time period.

Thus in this application all calls to web services will follow the following pattern: Activity→ Fragment→ Services→ Fragment→ Activity. The service that was created extends IntentService class in a new thread by calling the sent web service parameters:

```
if (request != null) {
    HttpClient client = new DefaultHttpClient();
    Log.d(TAG, "Executing request: "+ verbToString(verb) +": "+ action.toString());
    HttpResponse response = client.execute(request);
    HttpEntity responseEntity = response.getEntity();
    StatusLine responseStatus = response.getStatusLine();
    int statusCode = responseStatus != null ? responseStatus.getStatusCode() : 0;

    if (responseEntity != null) {
        Bundle resultData = new Bundle();
        resultData.putString(REST_RESULT, EntityUtils.toString(responseEntity));
        receiver.send(statusCode, resultData);
    }
    else {
        receiver.send(statusCode, null);
    }
}
```

To interact with the service, we created a fragment that will launch the service and interpret the results:

```
public abstract class RESTResponderFragment extends Fragment {

        private ResultReceiver mReceiver;

        public RESTResponderFragment() {
                mReceiver = new ResultReceiver(new Handler()) {

                @Override
                protected void onReceiveResult(int resultCode, Bundle resultData) {
                if (resultData != null && resultData.containsKey(RESTService.REST_RESULT)) {
                   onRESTResult(resultCode, resultData.getString(RESTService.REST_RESULT));
                } else {
                   onRESTResult(resultCode, null);
                }
              }};}

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setRetainInstance(true);
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
                return super.onCreateView(inflater, container, savedInstanceState);
        }

        public ResultReceiver getResultReceiver() {
```

DOI: 10.12948/issn14531305/17.2.2013.02

```
                return mReceiver;
        }

        abstract public void onRESTResult(int code, String result);
}
```

As we can see, we added a *ResultReceiver* variable to capture response offered by the service. For each module of the application, which should display data provided by the web service, we created a fragment extending the fragment described above. It must implement the method *onRESTResult* in order to interpret the received answer. For example, to display the list of reports available for the authenticated user, we developed a new fragment that executes the following method as soon as the activity has been created:

```
public void getReportsList() {
        ReportsActivity activity = (ReportsActivity) getActivity();

        if (activity != null) {
           dialog = new ProgressDialog(activity);
           dialog.setIndeterminate(true);
           dialog.setCancelable(false);
           dialog.setMessage("Please wait");
           dialog.show();
           Intent intent = new Intent(activity, RESTService.class);
           intent.setData(Uri.parse(MyHelper.SERVER_URL+"v1/reports/get_reports_list"));
           SharedPreferences settings = activity.getSharedPreferences(MyHelper.PREFS_NAME, 0);
           String prefUsername = settings.getString("username", "");
           String prefPassword = settings.getString("password", "");
           String prefUserType = settings.getString("user_type", "");
           Bundle params = new Bundle();
           params.putString("username", prefUsername);
           params.putString("password", prefPassword);
           params.putString("usertype", prefUserType);
           intent.putExtra(RESTService.EXTRA_HTTP_VERB, RESTService.POST);
           intent.putExtra(RESTService.EXTRA_PARAMS, params);
           intent.putExtra(RESTService.EXTRA_RESULT_RECEIVER, getResultReceiver());
           activity.startService(intent);
        }
}
```

To interpret the result we got *onRESTResult* method implemented as follows:

```
@Override
public void onRESTResult(int code, String result) {
        ReportsActivity activity = (ReportsActivity) getActivity();
        dialog.dismiss();
        if (activity != null && result != null) {
                ObjectMapper mapper = new ObjectMapper();
                ReportsApiResponse response = new ReportsApiResponse();
                try {
                        response = mapper.readValue(result, ReportsApiResponse.class);
                } catch (Exception e) {
                        Toast.makeText(activity, R.string.response_wrong_format,
                                        Toast.LENGTH_LONG).show();
                        e.printStackTrace();
                        activity.finish();
                        return;
                }
                // Check to see if we got an HTTP 200
                if (code == 200) {
                        reportsList = response.getReports();
                        activity.setReportAdapter(reportsList);
                } else {
                        Toast.makeText(activity, response.getMessage(),
                                  Toast.LENGTH_LONG).show();
                        activity.finish();
                }
        }
}
```

DOI: 10.12948/issn14531305/17.2.2013.02

Because until we get answers from the service on may take some time depending on the internet connection, we posted a message dialog that notifies the user that is running an operation that can last longer. For this we used a *ProgressDialog* with *setIndeterminate* property (true), which is displayed when the service is started. It is cleared when the service returns us an answer.

Web services provided by web application *YourView* are presented in JSON format. To interpret them within the mobile application, we chose to use the Jackson library that facilitates the transformation of a string in JSON format in a specified class. For example, a user authentication method returns a response like:

```
{
        "message": "You logged in successful-
ly",
        "type": "success",
        "result": {
                "user_type": "type",
                "companyid": "123456",
                "username": "test"
        }
}
```

To interpret it we created a class like this:

```
public class LoginApiResponse {
        private String message;
        private String type;
        private Result result;
        public static class Result {
                private String user_type;
                private int companyid;
                private String username;
                ...
        }
        ...
}
```

To convert the response received in the created class, we used the following lines of code:

```
ObjectMapper mapper = new ObjectMapper();
ReportsApiResponse response = new
ReportsApiResponse();
response = mapper.readValue(result,
ReportsApiResponse.class);
```

It is very important to take into account the response time of the application we create. Displaying a long list that can contain hundreds or even thousands of lines complicates the application's behavior, and whether the list is created by querying a web service then has to take into account the response time of the web service.

In the application *YourView* each report is displayed in a list form and can contain hundreds of lines. To take into account the above mentioned we proceeded as follows: first we need to create the list so that it can handle a large number of lines without hindering navigation list. The list is created custom: each line will display more details when the user clicks on it and for this we created a new class that extends BaseAdapter. Usually every list contains more items than are displayed on the screen and when the user navigates through the list, with rows also disappear the view sites associated with them. Objects that represent these lines can be reused for new rows displayed by *convertView* parameter of the method *getView()*, or for each line displayed should build Android layout xml file as:

```
public View getView(int position, View convertView, ViewGroup parent) {
        ViewHolder rowHolder;
    if (convertView == null) {
        convertView = mInflater.inflate(R.layout.report_alpha_list_item, null);
        rowHolder = new ViewHolder();
        rowHolder.rMainTitle = (TextView) convertView.findViewById(R.id.rMainTitle);
        rowHolder.rDetailsTop = (TextView) convertView.findViewById(R.id.rDetailsTop);
        rowHolder.rDetailsBottom = (TextView) convertView.findViewById(R.id.rDetailsBottom);
        rowHolder.rHiddenDetails = (LinearLayout) convertView.
              findViewById(R.id.rHiddenDetails);
        rowHolder.rMainRow = (RelativeLayout) convertView.
               findViewById(R.id.report_alpha_list_item);
        rowHolder.rToggleImage = (ImageView) convertView.findViewById(R.id.toggleRowImage);
        convertView.setTag(rowHolder);
    } else {
                rowHolder = (ViewHolder) convertView.getTag();
   }
...
```

DOI: 10.12948/issn14531305/17.2.2013.02

Through this approach the new elements are created much faster and browsing through the list is more natural. To solve the problem of the query web service and of displaying lines from the list in blocks of n-elements we had to customize the list:

```
public static class ReportList extends ListFragment implements OnScrollListener {

        private int currentPage = 0;
        private int previousTotal = 0;
        private boolean loading = true;

        @Override
        public void onActivityCreated(Bundle savedInstanceState) {
                super.onActivityCreated(savedInstanceState);
                getListView().addFooterView(pendingRow, null, false);
                setListAdapter(mAdapter);
                getListView().setOnScrollListener(this);
        }

        @Override
        public void onListItemClick(ListView l, View v, int position, long id) {
                ((ReportListAdapter) getListAdapter()).toggle(position);
        }

        @Override
        public void onScroll(AbsListView view, int firstVisibleItem,
                        int visibleItemCount, int totalItemCount) {
                if (loading) {
                        if (totalItemCount > previousTotal) {
                                loading = false;
                                previousTotal = totalItemCount;
                                currentPage++;
                        }
                }
                if (!loading &&firstVisibleItem+visibleItemCount>=totalItemCount &&
                                totalItemCount<totalRecords) {
                        if (getListView().getFooterViewsCount() == 0)
                                getListView().addFooterView(pendingRow, null, false);
                        reportResponder.getMessages(currentPage);
                        loading = true;
                }
        }

        @Override
        public void onScrollStateChanged(AbsListView view, int scrollState) {}
}
```

The web service may be configured to return a number of different results, depending on the ratio. This number is stored in the web application, so if you notice that a report generation time is large, we can adjust this number without having to release a new version of the application. The entire app was designed so flexible that most changes can be made directly from the web without requiring changes in the mobile application. So for each report, in the parameters those are sent to the web service is also included the page number to be displayed. This will give us the appropriate data set; for example, for a total demand of 50 results for page 1, the service will return the first 50 lines; to page 2 it will return results from 50 to 100, and so on. We store all these results in an *ArrayList* list, on which we create the list itself.

DOI: 10.12948/issn14531305/17.2.2013.02

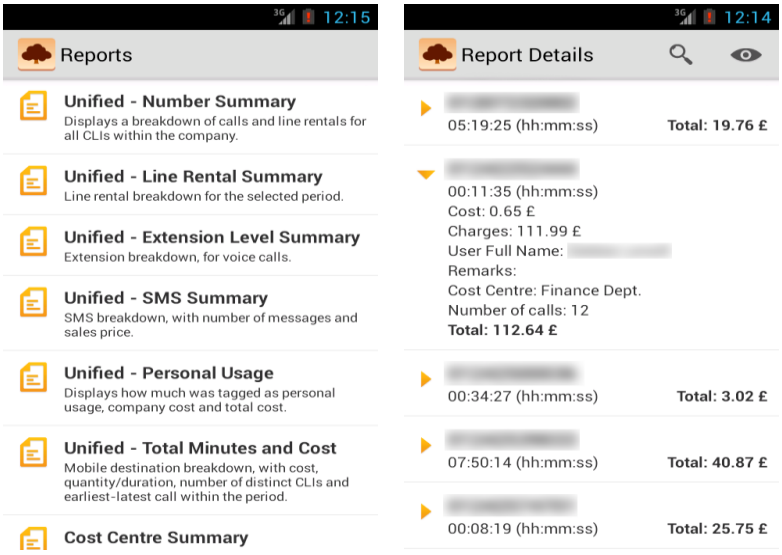

Fig. 4. The reports generated by the application

As you can see in the above code, we have implemented *OnScrollListener*, which will indicate which elements of our list are visible. When we reached the last item in the list, the application sends a request to the web service to bring us the next set of data. Once we receive the reply we add new items to the list and update variables.

The modern applications must meet a number of requirements [18] and we consider that our proposal is a valid one.

8 Conclusions

Modern mobile applications need a robust infrastructure and various services in order to obtain data from heterogeneous contexts. As we could see in this paper, REST web services are a valid option because they can be invoked in an elegant manner and they are really useful for the client application. Also, it is very important to synchronize the threads so that the application runs normally, without blocking sequences.

We consider that the future of SOAP and REST architectures will depend on the support offered by the industry in the implementation of new series of features according to the technological evolution.

References

[1] G.M. Tere, B.T. Jadhav, R.R. Mudholkar (2012). Dynamic Invocation of Web Services. *Advances in Computational Research*. ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 4, Issue 1, pp.-78-82.

[2] F. Felhi, J. Akaichi (2012). Adaptation of Web Services to the Context Based on Workflow: Approach for Self-Adaptation of Service-Oriented Architectures to the Context. *International Journal of Web & Semantic Technology (IJWesT) Vol.3, No.4 [OnLine]*. Available: http://airccse.org/journal/ijwest/papers/3412ijwest01.pdf

[3] A. N. Khan, S. Asghar, S. Fong (2011). Framework of integrated Semantic Web Services and Ontology Development for Telecommunication Industry. *Journal of Emerging Technologies in Web Intelligence*, Vol 3, No 2 (2011), pp. 110-119, May 2011

[4] C. Strimbei (2012). Smart Data Web Services. *Informatica Economica Journal*, Vol. 16, No 4, pp. 74-85

[5] B. Carminati, E. Ferrari, P.C.K. Hung (2005). Exploring Privacy Issues in Web Services Discovery Agencies. *IEEE Security & Privacy Magazine*, 3(5): 14-21, 2005.

[6] S. Khapre, D. Chandramohan (2011). Personalized Web Service Selection. *International Journal of Web & Semantic Technology Vol. 2, No. 2* [OnLine]. Available: http://airccse.org/journal/ijwest/papers/2211ijwest06.pdf

DOI: 10.12948/issn14531305/17.2.2013.02

[7] M. Champion, C. Ferris, E. Newcomer, D. Orchard (2012). Web Services Architecture. *W3C Working Draft* [Online]. Available: http://www.w3.org/TR/2002/WD-ws-arch-20021114

[8] C. Pautasso, O. Zimmermann and F. Leymann, "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision", *17th International World Wide Web Conference (WWW2008)*, 2008

[9] A. Cobârzan (2010). Consuming Web Services on Mobile Platforms. *Informatica Economicã Journal* [Online]. 14(3), pp. 98-105. Available: http://revistaie.ase.ro/content/55/1008%20-%20Cobarzan.pdf

[10] IBM Software Information Center (2011). The Structure of a SOAP Message [Online]. Available: http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=%2Fcom.ibm.cics.ts31.doc%2Fdfhws%2Fconcepts%2Fsoap%2Fdfhws_message.htm

[11] P. Glowacki (2012). Why Use "REST" Architecture for Web Services? [Online]. Available: http://edn.embarcadero.com/article/40467

[12] S. Tilkov (2007). A Brief Introduction to REST [Online]. Available: http://www.infoq.com/articles/rest-introduction

[13] S. Tyagi (2006). RESTful Web Services [Online]. *Oracle Technology Network*. Available: http://www.oracle.com/technetwork/articles/javase/index-137171.html

[14] L. Betianu (2012). "Indexes for a Sustainable Society", in *The Proceedings of the 6th International Conference on Globalization and Higher Education in Economics and Business Administration*, pp. 920-926

[15] D. Fotache, L. Hurbean (2006). Business Process Outsourcing. *Informatica Economica Journal*, Vol. 10, No. 2, pp. 49-54

[16] N. Goodman (2012). Modern Techniques for Implementing Rest Clients on Android 4.0 and Below – Part 2 [Online]. Available: http://neilgoodman.net/2012/01/01/modern-techniques-for-implementing-rest-clients-on-android-4-0-and-below-part-2/

[17] I. Ivan, A. Zamfiroiu (2011). Quality Analysis of Mobile Applications. *Informatica Economica Journal*, Vol. 15, No. 3, pp.136-152

[18] V.D. Pavaloaia (2009). Web based applications for SMEs economic and financial diagnose. *Communications of the IBIMA* No. 9, pp. 24-30

Octavian DOSPINESCU graduated the Faculty of Economics and Business Administration in 2000 and the Faculty of Informatics in 2001. He achieved the PhD in 2009 and he has published as author or co-author over 30 articles. He is author and co-author of 10 books and teaches as a lecturer in the Department of Information Systems of the Faculty of Economics and Business Administration, University Alexandru Ioan Cuza, Iasi. He is interested in mobile applications and enterprise integrated systems.

Marian PERCA graduated the Faculty of Economics and Business Administration and now he is a web developer for TheSoftwareFarm, UK. He is interested in mobile application development and is co-founder of the portal www.aplicatii-mobile.ro. He is co-author of a book and intends to develop new applications for financial services.

DOI: 10.12948/issn14531305/17.2.2013.02