Engineering Mobile Field Worker Applications

Pietro Mazzoleni
IBM T.J. Watson Research Center
Hawthorne, New York, USA
pmazzol@us.ibm.com

Stefan Tai
IBM T.J. Watson Research Center
Hawthorne, New York, USA
stai@us.ibm.com

ABSTRACT

We envision next-generation Field Workers to use interactive Web applications deployed on sophisticated mobile devices in order to access remote services and data in support of their business processes. Engineering such applications requires combining services computing with mobile computing and end-user oriented Web application development. In this position paper, we identify the main challenges in this context, and propose a preliminary platform architecture for application deployment on Field Worker devices that addresses some of these challenges.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Field Workers, Mobile Services

1. INTRODUCTION

Services departments in Automotive, Manufacturing, Electronics and many other industries require "anyplace" access to relevant technical information. Anyplace data access across heterogeneous and changing network environments ensures that complex tasks can be resolved on-the-spot and when needed, promising an increase in productivity and customer satisfaction. For this reason, companies rely on "Field Workers" – mobile experts that work on requests from various locations.

While this "mobile force revolution" started many years ago (with the arrival of laptop computers) [6] recent advances in technology and Web computing introduce new opportunities:

- *Services computing.* As service-oriented computing technologies mature and gain acceptance [8], the number of (Web) services accessible by Field Workers increases. Further, as computers are increasingly embedded in the machines that Field Workers are asked to fix or maintain [3], services provided by the machines become available. Field Workers must be able to access all kinds of services across diverse pervasive environments and Beyond 3G (B3G) networks. In the Automotive domain, for example, Field Workers may use both Web-accessible and car services to execute diagnostic Engine procedures at a Dealership location.
- *Resource "unconstrained" mobile devices.* Next-generation mobile devices are expected to be (nearly) as powerful as desktop computers. Today, we can already assume Field Workers to be equipped with small, lightweight but powerful mobile devices such as Micro PCs [1].
- *End-user-oriented Web applications.* As the general population is becoming more and more familiar with the Internet and the use of Web applications such as Web 2.0 digital communities, data exchange sites, and social networks, we expect professionals including Field Workers to expect and benefit from applications that offer similar features and follow a similar interaction paradigm.

Another important consideration in the engineering of Field Worker applications is the emergence of *Services-led Businesses.* In the past, Field Workers were typically directly hired by a manufacturer; today, manufacturers increasingly rely on contracted Field Workers. In the latter case, a single Field Worker may then work for multiple manufacturers, with each manufacturer requiring its own set of processes and standards to be followed, and thus its own set of services and applications to be used. Further, Field Workers may be *micro-entities* (a single or a few person business) and thus demand applications in support of very small businesses, for example, applications that simplify and ease customer collaboration, contracting, and pricing.

In this paper, we discuss the main challenges in engineering next-generation Field Worker applications and present a platform architecture specifically designed in consideration of the above described opportunities. Our approach is based on a *Service Mediator* component that seamlessly accesses any kind of service (either on the Web or on B3G networks). We further introduce *SOA Widgets* as a way to manage a variety of Field Worker application interfaces and services interaction patterns.

[1]A typical Micro PC today runs Windows, has a large HD, up to 1Gb of Ram, has a screen size of about 5-6 inches, weights less than 1.4 lbs (600 grams), and can be wireless connected to both GPS and GPRS networks.

ESSPE '07 September 4, 2007, Dubrovnik, Croatia

2. OPEN CHALLENGES

We can identify as the main challenges in engineering next-generation Field Worker applications:

Uniform access to ubiquitous Services: Field applications have to access and compose diverse kinds of Web services and data (SOAP, REST, Atom, RSS feeds) that may be deployed on enterprise servers, be available in B3G networks, or be deployed on other mobile devices. A uniform programming and runtime model to access these services is needed.

Context-Aware Service development: Even though the notion and use of contexts have been investigated intensively for mobile applications [1, 2], creating a Web Service whose behavior depends on contextual information is still a tedious and complex task [4, 5]. For Field Worker applications, we are less concerned about resource constraints imposed by different mobile devices, but are interested in contextual information such as physical location and business context.

Consider, for example, the task of selecting a Field Worker from a set of available workers and assigning him for a pending job. The semantics of this process can be quite complex and contextual information such as physical location and workload of the workers, customer (priority) status, severity of the issue to be solved, etc may be critical for the decision making. While there are methods and tools for implementing standard Web Services (for example, given a WSDL specification, stubs and skeletons are generated), there are no methods and tools available in support of context-aware Web Services development.

Data management and visualization: Field Workers require access to potentially very large sets of data. Thus, there is a need for implementing effective means for data retrieval and visualization. For example, large data may be retrieved via asynchronous messaging, and RFID and GPS technologies are two examples of contextual information that can serve as filters for data to be presented.

Again, Field Workers are expected to execute potentially complex tasks using their mobile device (such as executing a set of diagnostic procedures to discover the root of a problem in a car), and thus require appropriate interfaces for presentation and services interaction.

Mobile Web 2.0: Another important challenge relates to the use and practice of Web 2.0 technologies [7], and in particular the creation of dynamic Field Worker networks and communities for service and data exchange (for example, communities of all workers in a specific geographic region or of workers with specific skills) [9]. Applying Web 2.0 to mobile applications promises to advance communication and collaboration among workers, and enable community-based knowledge sharing and joint issue resolution.

3. CLIENT PLATFORM ARCHITECTURE

To address some of the key challenges listed in the previous Section, we now introduce a client platform architecture we are building to support the development and deployment of Field Worker applications. Our solution applies in general to any mobile application requiring intensive user interaction and access to diverse data and functional services.

The architecture is going to be developed in the context of the PLASTIC project[2], a European research project that addresses the development and deployment of cost-effective mobile application services across B3G networks.

The client platform architecture is illustrated in Figure 1; the Figure illustrates the components that are installed on the mobile device. At the server side, we only assume Web Services to be available.

At the presentation layer, the *SOA-Widgets* represent the user interface elements interacting with the services in the network. SOA-Widgets make use of modern Web technologies (such as asynchronous Javascript [3]), but, different from existing solutions, they are specifically built to simplify services interactions through XML specifications. More details on SOA-Widgets will be provided in Section 4.

At the middleware layer, the *SOA-Widgets Interpreter* analyzes the SOA-Widgets XML specifications and based on events generated by the user executes one or more of the following actions: running some program (through the *Widget Behavior* component); executing a BPEL process (through the *BPEL Engine* component), or invoking one or more services (through the *Service Mediator* component).

The Service Mediator component is the core element of the middleware layer. The mediator is in charge of mediating the interaction with all services needed by the application; authentication, system transactions, and managed access to services available on B3G networks are examples of mediating functions that can be supported.

Existing middleware such as Apache Synapse[4] (a distributed service mediation framework based on Web services, XML, and Axis2) can be used to implement the Service Mediator. In our Field Worker context, we foresee the need to extend existing mediator technology such as Apache Synapse to mediate requests to services on B3G networks. Here, the mediator needs to convert requests from the interfaces supported by Synapse (WS-* and RESTstyles) to the ones supported by B3G services and vice-versa.

Finally, at the data source level, we include any services deployed on the mobile device. Two types of such services exist: *Context-Services*, which expose contextual information as discussed earlier (e.g., physical location), and *Functional-Services*, which provide specific functions needed by the application (e.g., a Field Worker calendar and agenda service).

Our client architecture promises to introduce a number of advantages. First, SOA-Widgets, designed for interactive service-oriented architectures, promote the reuse of interface elements across various mobile applications.

Second, the Service Mediator is designed to enable and guarantee uniform access to any Internet and B3G service. Third, the Widgets interpreter can significantly reduce the effort needed by an application developer to write code for managing services interactions. The idea is to define the rules for an open library of visual components that everybody can contribute to, and to make available an engine which can interpret and compose the components based on the Services that each mobile application requires.

[2]http://www.ist-plastic.org/

[3]In this paper, we intentionally omit the term Ajax because the programming model used by Ajax is only one of the possible models that can be used to implement SOA-Widgets

[4]Apache Synapse is an Apache Incubator project available on http://ws.apache.org/synapse/

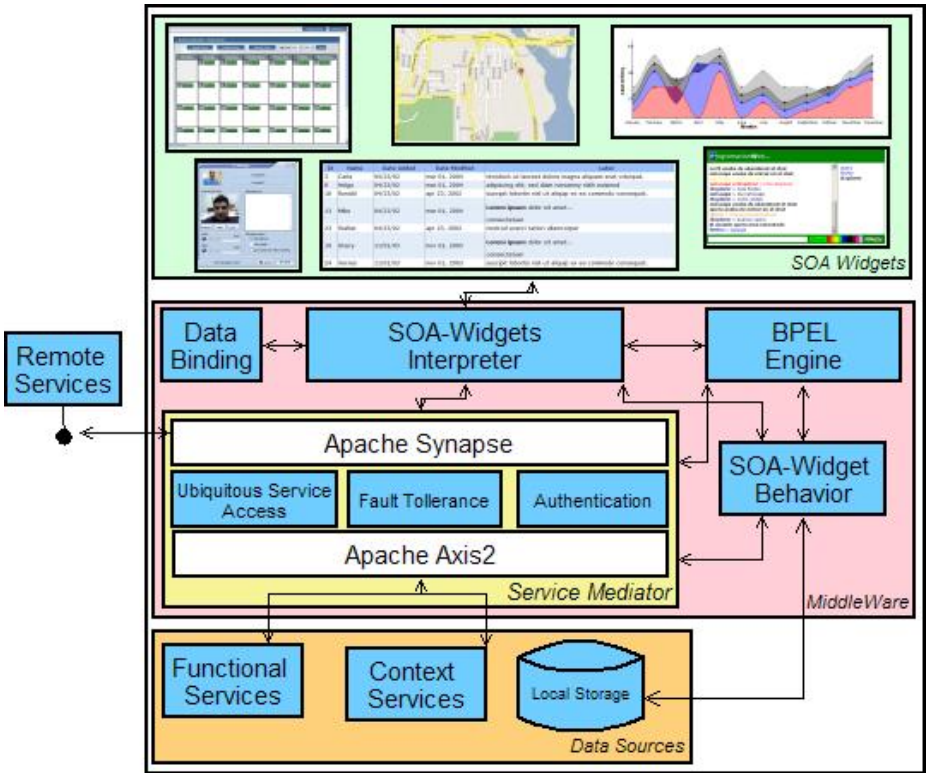

Figure 1: System Architecture

4. SOA-WIDGETS

Next, we discuss in more detail SOA-Widgets, which are the user interface elements designed for services interaction.

SOA-Widgets are similar to other visual components which are available today on the Web[5]. However, SOA-Widgets are not just another set of visual components, but rather an approach that can be applied on top of existing Widgets to interact with Web Services. The approach is quite straightforward: each SOA-Widget is built as a template for a specific XML schema element (the complexity of which depends on the Widget). An XML specification is used to describe how the SOA-Widget schema can be bound to the WSDL types used by one or more Web Services. The XML specification further contains information for associating events to the Widget and to execute one or more of the following tasks: to invoke a Web Service (and recursively render other SOA-Widgets), to run a program, or to execute a BPEL script.

Figure 2 and Figure 3 show a simple example of an XML specification for SOA-Widgets. In the example, a Widget is used to present data of all gasoline stations in a region[6]. In addition, an event has to be associated to the Widget such that when the user selects one of the entries, another Service is invoked to retrieve (and present in another Widget) the number of issues pending at the selected gasoline station.

In Figure 2, a Widget called *GasTable* is defined. The Widget makes use of WSDL2Table, a SOA-Widget we created starting from the Dojo FilteringTable Widget[7]. WSDL2Table takes as input any sequence of elements and arranges them in a sortable XML-spreadsheet. In the specification, a Widget is represented by three elements: <input>, to identify the source of the data to be displayed, <appearance> to describe how such data should be displayed, and *event* to list the events supported by the Widget. In our example, the input for the *GasTable* Widget is the result of the invocation of the *GasStationWS* Service. The appearance of the Widget is derived from the WSDL type Supplier of the same Service (further customization have been omitted for the sake of readability) and the field <htmlContainer> specifies the location (HTML container) in the Web page where the output should be rendered. Finally, a "Click-button" event is associated to the Widget such that when triggered, the Service called *PendingIssueWS* will be invoked.

The XML specification describing the *GasStationWS* and *PendingIssueWS* services is illustrated in Figure 3. Each Service is described by an ID, and EPR (EndPoint Reference), and a method. Similarly to the Widget specification, the input of the Service can be either statically specified in the specification file (as for *GasStationWS*) or dynamically collected as a result of an external event such as the ones associated to the Widgets (as for *PendingIssueWS*).

Once the Service has been invoked, its results can be rendered to multiple Widgets (in addition to the one triggering the invocation). This is the case of the *PendingIssueWS* Service, the results of which are rendered on a different Widget, called *NumIssuesWidget* the definition of which will be sim-

[5]Dojo, Rico, Jquery, and Yahoo Widgets are examples of frameworks offering extensible sets of Widgets to perform different tasks.

[6]The data for the example is available from commercial services such as the StrikeIron Gasoline Station Service accessible at http://www.strikeiron.com/ProductDetail.aspx?p=279

[7]http://dojotoolkit.org

```
<!- - List of Widgets participating in the page- ->
<Widget id="GasTable">
<WidgetTemplate>WSDL2Table</WidgetTemplate>
  <input>
    <!- - the Widget can be immediately rendered –>
    <xml-text/>
    <!- - Services to be invoked before renderering the Widget- ->
    <service-id>GasStationWS</service-id>
  </input>
  <appearance>
    <HtmlContainer>Area04</HtmlContainer>
    <Display>
    <!- - This complextype is the one returned from the WS and it can be automatically extracted from the WSDL - ->
      <complexType name="Supplier"/>
  </appearance>
  <events>
    <InternalEvents/>
    <ExternalEvents>
      <event Id="click-butt">
        <function> <name>SingleData</name> <parameters>State</parameters> </function>
        <serviceId>PendingIssueWS</serviceId>
        <BindingFunct>
          <name>SingleBindGasToServiceIssue</name> <parameters/>
        </BindingFunct>
      </event>
    </ExternalEvents>
</events>
</Widget>
```

Figure 2: Widget Specification

ilar to the one described for *GasTable*.

No additional tasks are required to model our simple example. The Widget Interpreter takes the specification, prepares the service invocation (after executing programs and/or running BPEL processes described in the specification), and submits it to the Service Mediator. The Service Mediator will then discover (eventually on B3G networks) the actual service to be invoked, will convert the request (if needed) to the interface supported by the discovered service, and will invoke it. The results are then collected by the Widget Interpreter which will decide the program to execute, the BPEL process to run, and new Widgets to be displayed with their content and supported events.

The relation between SOA Widgets and Services is N:N. One Widget can be used to simultaneously render multiple Services while a Service can be rendered on multiple Widgets. The only restriction is the nature of the Widgets and the XML schema (and set of events) it supports.

We consider the use of SOA-Widgets to be important for Field Worker applications. As discussed in Section 2, Field Workers need access to many data sources and must execute many tasks on the machines to be fixed. For this reason, a library of SOA-Widgets specifically built to support Field Workers activities will be provided as part of our solution. Among others, we are considering SOA-Widgets to render both proximity (through RFID) and geographical (through GPS) information, agenda management, charts, and community elements such as chats and knowledge forums. In addition, specific Widgets will be created to manage Diagnostic and Resolution protocols which are the tasks needed to solve a technical issue. The protocol SOA-Widgets will contain several elements which are customizable according to services with which they will interact. Specifically, the protocol Widget considers the following aspects:

- Organization of the process flow structure considering both sequences, alternatives, and cycles.
- Identification (through proximity) of the various components and use of geographical location information to support parts pricing and availability.
- Interaction with diagnostic instruments (if available) and logging of testing procedures performed on the machines.
- Automatic management of time consumed and overall cost calculation.
- Management of relationships among multiple repair protocols sharing similar diagnostic procedures and issues to be solved.

In our approach, the library of SOA-Widgets will not be proprietary but rather open for contributions. Developers can make available their components to the community and use the ones shared by others. This approach has proven useful for implementing visual components (e.g.www.dojo.com) but has not been used for Services components for mobile applications.

From a technical point of view, the SOA-Widget Interpreter has been written completely in Javascript which makes it usable independently from the framework and the programming model used to create the SOA-Widgets. With this approach, the advances (and the new Widgets) of frameworks such as Dojo, Rico, and Jquery could be easily integrated with the mobile application.
Notice, the decision to implement everything through SOA-Widgets or to use them only for Service interaction (and implementing the rest using HTML or any server-based scripting language) is left to the application developer. SOA-Widgets (and its engine) are built as Web components, and therefore they can be used in different ways. Our client platform architecture however readily supports SOA-Widget-based application development.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we described how advances in services computing and Internet technology together with changes in business models for Field Workers are promoting a new

```
<service id="GasStationWS">
   <!- - Information concerning the service to be invoked- ->
   <epr>http://localhost:8008/axis2/services/GasStationWS/</epr>
   <operation>http://www.strikeiron.com/GetSuppliers</operation>
   <namespace>http://www.strikeiron.com/</namespace>
   <input/>
      <xml-text>
         <ns1:GetSuppliers> <ns1:ZipCode>89104</ns1:ZipCode> </ns1:GetSuppliers>
      </xml-text>
   <output>
      <WidgetId>MyLocationContext</WidgetId>
   </output>
</service>
<service id="PendingIssueWS">
   <!- - Information concerning the service to be invoked- ->
   <epr>http:// localhost:8008/axis2/services/GetPendingProxy/</epr>
   <operation>GetPendingIssue</operation>
   <namespace>http://www.manufacturer.com/</namespace>
   <input/>
   <output>
      <WidgetId>NumIssuesWidget</WidgetId>
   </output>
</service>
```

Figure 3: Service Specification

phase of the "mobile force revolution". Different from the past, where most challenges were related to managing offline activities and executing programs on "resource-constrained" devices, today a number of other, additional challenges are emerging. Solutions are needed which combine Services Computing with mobile B3G network computing and enhanced user interface elements and end-user orientation. To address these challenges, we presented a platform architecture including the SOA-Widget model which simplifies the process of engineering mobile services applications.

The work presented in the paper is work-in-progress. As a next step, a more complete formalism for SOA-Widgets will be prepared and the SOA-Widget Interpreter will be made available for use and testing. In addition, the architecture presented in Figure 1 will be refined taking into account the work done by other partners of the PLASTIC project. Finally, we aim to improve the development of context-aware services by extending the well-known WSDL2Java tool to include contextual information.

6. ACKNOWLEDGMENTS

This work is partially supported by the PLASTIC Project (EU FP6 STREP n.26955): Providing Lightweight and Adaptable Service Technology for pervasive Information and Communication. http://www.ist-plastic.org.

7. REFERENCES

[1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, pages Vol. 2, No.4 pp. 263 – 277, 2007.

[2] A. K. Dey. *Providing architectural support for building context-aware applications*. PhD thesis, 2000.

[3] S. Jin. Ubiquitous service-oriented architrecture. *W3C Ubiquitous Web Workshop.*, 2006.

[4] P. R. Julien Pauty, Davy Preuveneers and Y. Berbers. Research challenges in mobile and context-aware service development. In *Proceedings of Future Research Challenges in Software and Services (FRCSS 06)*, 2006.

[5] Z. Maamar, D. Benslimane, and N. C. Narendra. What can context do for web services? *Commun. ACM*, 49(12):98–103, 2006.

[6] E. NewComer. Mobile orchestration: Using web services to empower your mobile workforce. *Web Service News*, 2003.

[7] T. O'Reilly. What is web 2.0? design patterns and business models for the next generation of software. 2005.

[8] J. Spohrer and D. Riecken. Introduction. *Commun. ACM*, 49(7):30–32, 2006.

[9] S. Tai, N. Desai, and P. Mazzoleni. Service communities: applications and middleware. In *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, pages 17–22, New York, NY, USA, 2006. ACM Press.