Applying SOA Principles on Mobile Platforms

Martin Treiber
Distributed Systems Group
Argentinierstrasse 8/184-1
1040 Wien, Austria
ttreiber@infosys.tuwien.ac.at

Christian Scherling
ikangai solutions
Treustrasse 59/5/20
1220, Wien, Austria
office@ikangai.com

Schahram Dustdar
Distributed Systems Group
Argentinierstrasse 8/184-1
1040 Wien, Austria
dustdar@infosys.tuwien.ac.at

ABSTRACT
Principles of Service Oriented Architectures are well adopted for realizing complex distributed Service-based applications (SBA). In the mobile domain, however, we observe a different trend. App Stores emerged as main distribution channel and small standalone Applications (Apps) are the main mean to distribute software. We study the applicability of SOA principles for the mobile domain with respect to existing infrastructures like App Stores. We present a software framework that uses adapted SOA principles to provide the means to construct App-based Applications while following SOA principles. Taking mobility into account, we explain how our proposed software framework works with physical objects tagging and we illustrate the use of App compositions that are bound to physical objects. Software Architectures

Categories and Subject Descriptors
D.2.11 [Software Architectures]: Service-oriented architecture (SOA)

General Terms
App Based Applications

Keywords
Mobile SOA, Apps, Physical Object Tagging, QR Barcodes, App Store

1. INTRODUCTION
SOA principles [3] have become a predominant way of implementing complex software architectures in the business world [40]. Web Services are the most common way of implementing Services because of their well adopted standards; the WS-* stack defines key aspects of a Web Service, with WDSL [6] to describe the Web Service interface or WS-Addressing [5] to specify the address of a Web Service. Consequently, there are a numerous software frameworks, Web Service containers respectively, that implement the WS-* stack like Apache AXIS2 1 Glassfish2.

At a first sight, in the mobile domain, SOA principles seem less adopted [30]. While there are several implementations of Web Services for mobile devices, their main focus is on the Service consumer side and not on the Service provider side. The Web Service stack is generally considered too heavy-weight for mobile devices due to their resource limitations (cpu, memory, connectivity) and available implementations must adapt their features and limit their features to a subset of the WS-* stack like gSOAP 3.

With the arrival of mobile operating systems like Android or iOS a different way of providing Services to their users emerged. Generally speaking, Apps (small independent pieces of software with limited functionality) are the means to add functionality, Services respectively, to mobile platforms. The distribution of Apps is done by a centralized App Stores like the Android Store or the iTunes App Store. We argue that Apps and App Stores can play central roles for SOA in the mobile domain. In particular, App Stores can be viewed as registries for Apps, which can be retrieved with basic discovery mechanisms based on keywords. Thus, App Stores play the role that Web Service registries (e.g., UDDI [7, [23]) supposed to: a centralized and universal registry for Services. By following this approach, we argue that Apps can be considered as Services themselves and that the binding process of Apps resembles conceptually Service binding.

However, there are limitations due to the nature of Apps. Apps are typically developed independently of each other and do not offer standardized interfaces like Web Services. Furthermore, Apps cannot be composed like Service Based Applications. With this limitations in mind, we elaborate on the differences and similarities of Services and Apps in the following two sections, where we investigate the role of App Stores and Apps in the light of SOA principles. In the following section 4 we introduce our concept of creating App bases Applications and introduce a grammar to specify App compositions. We continue in section 5 with a discussion on the App binding process for App compositions. Then,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 2010 Waikiki, Honolulu, Hawaii, USA
Copyright 20XX ACM X-XXXX-XX-X/XX/XX ...$10.00.

1http://ws.apache.org/axis2/
2https://glassfish.dev.java.net/
3http://www.cs.fsu.edu/~engelen/soap.html
we explain how we apply our concept on physical objects in order to attach App compositions to physical objects. In section 7 we describe of our proposed architecture and details of the prototype implementation. We conclude the paper with related work and an outlook for future work.

## 2. APP AS A SERVICE

In the Service world, the term software as Service (SaaS) is referred to software on demand that uses Internet Technologies to distribute the software to customers [35]. If we apply the same concept to Apps, we can observe similarities. Both, Apps and SaaS are executed on thin clients. Services like Apps can be discovered with the help of centralized registries. From a Service consumer’s perspective there are no notable differences. However, there are three conceptual and technical differences between Services (as a Service) and Apps that need to be kept in mind when applying Service principles on Apps.

First of all, Apps lack standardized interface descriptions. Apps Stores do not offer any description of interfaces like WSDL for Apps. This is a consequence of Apps being small, independent applications which do not to communicate with other Apps. Instead, Apps provide rich user interfaces for the direct interaction with the end user. To address the limitation of inter App communication, we argue that URI Schemata can be used to transfer data between Apps. We propose the use of URI schemata to register Apps locally with certain types of URLs. Android and iOS come with a set of URI schemata which include standard URI schema like sms: to invoke the SMS app, tel: to make phone calls or http: to invoke Web Browsers on the device. Listing 1 shows an standard URI schema for an App (qLaunchert) that is capable to parse the data and to pass the data to another App (qCARD).

```
data:application/xml; charset=utf-8, com.ikanagai.qlauncher,
<q1>
  <id>qCARD</id>
  <u>
    <t>0042622033</t>
  </u>
</q1>
```

Listing 1: URL schema example

It is worth noticing that in order to invoke the App via URI schema, the accepted data format must be known a priori. Unless the URI schema is a standardized by bodies like the IETF, there is currently no possibility to determine the supported data format of URL registered Apps. We elaborate on the issue of URI schema registration in section 5 where we discuss the implications of custom URI schema registration.

Second of all, Apps are installed on the mobile device and not accessed remotely. Thus, the user of the App has direct control over the App and governs the execution, unless the App acts as interface to remote Services (e.g., RSS newsreader). This is in contrast to Web Services which are accessed remotely. With Web Services, users have no direct influence on the execution. To enforce Web Service behavior, SLA [16] have emerged which are used to control Service execution with regard to the conformance of pre-defined QoS criteria.

And finally, Apps have graphical and haptic user interfaces that be directly accessed by the user on the screen of the mobile device. Thus, the user interacts directly with the App. Unless Web Services are not packaged into Applications with user interfaces, Web Services do not provide graphical interfaces for direct user interaction.

We summarize our findings in table 1 which gives an overview of the properties of Services and Apps.

### Table 1: Comparison of Apps and Services

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>App</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locality</td>
<td>Local</td>
<td>Remove</td>
</tr>
<tr>
<td>Governance</td>
<td>Yes, direct</td>
<td>indirect, SLA</td>
</tr>
<tr>
<td>Interface Description</td>
<td>No</td>
<td>WSDL, WADL</td>
</tr>
<tr>
<td>Data Exchange</td>
<td>URL Schema</td>
<td>SOAP</td>
</tr>
<tr>
<td>Endpoint</td>
<td>URL Schema</td>
<td>WS-Addressing</td>
</tr>
<tr>
<td>Graphical User Interface</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

3. APP STORE AS A REGISTRY

The original purpose of Web Service registries [10] like UDDI [7] was to create a universal and centralized registry for services. As history showed, this didn’t work as envisioned. IBM, Microsoft and SAP abandoned their public UDDI registries in 2006 [21]. The actual reasons for the failure of UDDI becoming the universal registry are manifold. One of the reasons appeared to be the lack of available Web Services in UDDI registries. Investigations showed that there were 100s of Test Web Services registered in public registries. In addition, most Web Services offered no detailed description besides WSDLs and links to Web pages of the Web Service provider [2]. Even if Web Service descriptions were available, the corresponding Web Services were sometimes not available.

Until recently, with the exception of Seekda (Figure 2), there were no public registries available that offered a considerable set publicly available Services for customers. This limited the use of registries to niches where custom solutions were built and used for scientific investigations (e.g., AWSR [34] or [20]) and not for a broader public use.

Another promise of Service registries and of UDDI in particular, was to support automated Service discovery [19] and thus support for dynamic binding of Services [9] for Service Based Applications. UDDI and other registry implementations offered a set of (semantic) meta data for that purpose [1]. However, the adaption of semantic meta data for Service discovery [27] never took off and remained subject to academic prototypes and not wide spread use.

We argue that some of this limitations have been overcome by App Stores. App Stores play the role of registries in the mobile domain, albeit in a slightly different manner than registries. Like registries, App Stores are centralized, well known locations to discover Apps. In contrast to Web Service registries, access to the App Store is realized by a dedicated App Store App on the device, making the Service consumer implicitly aware of the location of the App Store.

However, the discovery process in App Store is offers limited capabilities in comparison with Web Service registries. App Stores typically provide a pre-defined set of categories that classify the Apps in a broad sense. Unlike Web Service registries, App Stores do not provide mechanisms to include semantic meta information to discover Apps automatically and rely purely on human judgement during the App selection process.

For example, the iTunes App Store offers 20 Categories (e.g., Productivity, Entertainment, News) and does not include sub categories for further refinement of the App discovery. The exception are recommendations of the iTunes Store (e.g., new and noteworthy, staff picks, top selling, top grossing) and the ability to sort the presented Apps according their popularity or creation date. In addition, keyword based discovery is provided by iTunes Store. Meta information about the Apps is provided as human readable text that is a mix of advertisement and feature description (see Figure 1).

The Android Store provides the same basic features as the iTunes store, but differs in the App admission process. In contrast to the iTunes Store, Apps in the Android store are not subject to a review process and can be uploaded without the permission of Google.

Another difference to Web Service registries is the fact, that App Stores actually provide the App itself. Customers can download and install the App directly from the App Store. Generally speaking, Web Service registries provide links to Services and typically do not offer access to the Services itself and offer no guarantees of existence of Web Services whereas App Stores can guarantee the existence of the App. Complementary to Apps stores, Web portals exist that access the App Store data. These portals offer links to the Apps and offer additional information like community ratings, App rankings of reviewers or App reviews to browse trough. In principle, this kind of portals can be enriched with semantic meta data and thus support automated App discovery. However, to the best of our knowledge none of the portals that provide links to Apps in App Stores offer semantic meta information.

We summarize the discussion in table 2 where provide an overview of Registry and App Store characteristics.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>App Store</th>
<th>Service Registry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery mechanisms</td>
<td>Manual</td>
<td>Automated</td>
</tr>
<tr>
<td>Meta Data</td>
<td>Human readable descriptions, Pre defined App Categories</td>
<td>Interface descriptions, Ontologies</td>
</tr>
<tr>
<td>Well Known Address</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic Binding</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Interface Descriptions</td>
<td>None</td>
<td>Yes, different standards</td>
</tr>
<tr>
<td>Human readable information</td>
<td>Yes</td>
<td>Yes, partially</td>
</tr>
<tr>
<td>Automated Discovery</td>
<td>No</td>
<td>Yes, supported</td>
</tr>
</tbody>
</table>

Table 2: Comparison of App Stores and Service Registries

4. BUILDING APP BASED APPLICATIONS

A central component of Service Oriented Architectures are languages which are used to construct Service based Applications (SBA). Examples are complex workflow languages...
like BPEL [41] or YAWL [36] which specify which services SBAs are used and how they collaborate. On mobile operating systems like Android or iOS, no dedicated language is available to compose Apps from existing Apps. As already stated, the reason is that Apps are considered to work independently of each other and serve a dedicated purpose. For example, an App that calculates meters from yards is a priori not able to invoke another App with this data. This requires manual activities of the user, like copying text into a clipboard and pasting later from the clipboard to another App.

In order to bootstrap the composition of Apps, we propose a simple grammar that allows for the composition of App Based Applications. In the current version of our proposed grammar, we support sequences of App invocations and the passing data of data between Apps. We include the capability to read context information and to enrich data passed to Apps with geo data, language settings and the passing data of data between Apps. We include the possibility to bind Services at runtime. This allows to dynamically bind an App to the custom URI schema that is the foundation for a later creation of context aware mobile App compositions.

To make the App composition work, Apps are also required to post a notification when they are closed or become inactive, and to write the output data into shared memory. This requires developer to modify the source of the App according to the data that needs to be shared. Listing 2 shows an example of that copies data into the clipboard after a user selected a contact from the address book in an iPhone App.

![Figure 3: Grammar for constructing App bases Applications](image)

Listing 2: Coping Data into Shared Memory

We now show how we compose Apps to a new, App Based App with our proposed grammar. Two Apps (q:card and mail) are invoked in a sequence and read context information (location and user language settings) as shown in Listing 3.

```plaintext
Listing 3: App Composition Example
The execution of the workflow starts with the invocation of q:card. If the user selects a contact in q:card from the address book, contact information is copied into the clipboard (as shown in Listing 4).

Listing 4: qcard data example
After terminating q:card, the telephone number is extracted (using regular expressions) from the shared data and the mail App is started. The mail App receives the geo location, the extracted email address as input. To the end user the App composition presents itself as shown in Figure 4.

5. APP BINDING WITH URI SCHEMATA

The binding process of Service Based Applications offers the possibility to bind Services at runtime. This allows to create Service Bases Apps that can react to context changes and select Services accordingly for a later invocation. The binding of Apps is similar and allows also the binding of Apps during runtime. However, there is one major difference to binding of Services. In contrast to Services, Apps do not have an explicit endpoint which can be used for binding. To address this limitation, we utilize URI schema [4] of mobile operating systems like Android or iOS to bind Apps. The binding however, needs to be made during design time and cannot be changed later. Listing 5 shows an example of the binding of and App to the custom URI schema qCARD:

```
1. Read URI schema and search local registry for App that architecture (see section 7) and includes the following steps:

The binding process of Apps is tailored to our proposed

If URI schema cannot be found in local registry, connect

2. If an App can be found, ask user to download and install App.

2.a. If no App can be found, ask the user to specify an App that is able to handle the URI schema.

2.b. If no App can be found, as the user to specify an App that is able to handle the URI schema.

3. If URI schema can be found in local registry, then determine if App is installed on mobile device

3.a. If App is installed, execute App and pass data to the App

3.b. If App is not installed, ask user to download and install App

The use of custom URI schema for binding Apps has several implications. The reason for the of use custom URI schemata is the lack of support to invoke Apps programmatically on mobile operating systems like iOS and Android. Current versions of these operating systems simply do not offer the functionality to identify handles to Apps programmatically. Instead, URI schema are used to get handles to installed Apps and to invoke them programmatically. To overcome this limitation, we propose to use of the data URI schema [32] as container to embed information for the invocation of Apps. This requires some additional considerations for the general use of the data: URI schema. Firstly, we require the presence of an App identifier, other than the custom URI schema. A candidate for this is the bundle identifier which is needed to identify the App bundle as shown in 6.

Listing 6: App Bundle Identifier

Secondly, we need to adapt the <data> part of the data: URI schema to include the identifier of the App that should be called with the data. A straightforward solution is to include the App identifier at the beginning of the data part, to identify the App that should handle the data. We show an example in Listing 7.

Listing 7: URL schema binding to App

However, the proposed approach need direct support by the mobile operating system, which is not available in the current versions of iOS and Android. We discuss how to address this limitation and propose an alternative implementation that we discuss in detail in section 7.

6. ATTACHING PROCESSES TO PHYSICAL OBJECTS

We discussed central concepts of SOA (services, registries, binding and compositions) and their application (and modifications) to the mobile domain in the previous sections. In this section, we extend the principles we’ve established in the previous sections with regard to the physical context of mobile devices. We introduce the concept of adding processes, App compositions respectively, to tangible objects. We borrow the concept of object hyperlinking [17] which creates a link between a physical objects and their virtual representation [18].

Our main idea is to QR barcodes [15] attached to objects as entry points for processes which are executed on the mobile device. We use our grammar introduced in section to compose Apps and encode the compositions in QR barcodes. By tagging a physical object with QR barcode, we implicitly add the process, App composition respectively, to a physical object (see Figure 5). This allows us to create novel applications, which are strongly embedded in the physical context. For example, a QR barcode that contains a process which includes an information App about a taxi company (e.g., a description of the Services offered like special rates for airport transfers) is placed on a taxi stand. Another App of the composition is used to contact the taxi company and this App can be automatically enriched with contextual information (GPS location, time, language settings). The taxi company could then make a reservation for a taxi which driver speaks the language of the customer. The

Figure 4: Screenshots of workflow execution

Listing 5: URI schema binding to App

The binding process of Apps is tailored to our proposed architecture (see section 7) and includes the following steps:

1. User selects contact in q:card

2. Contact is copied into clipboard

3. Mail App is opened with mail address

4. Geolocation is added to eMail
example shows how the physical context is integrated into
the App composition. On a conceptual level, we provide an
active extension to a physical object when attaching Apps,
App compositions that are embedded in QR barcodes. In
contrast to existing approaches which typically add URLs
to a physical object, we trigger the execution of a process
and thus implicitly add user interactions to physical objects.
This narrows the gap between of objects and activities which
happen in the virtual extension of the physical object.

7. ARCHITECTURE AND IMPLEMENTATION

Our proposed architecture addresses some of challenges
we discussed in the previous sections (see Figure 6). In or-
der to cope with the lack of meta data, we propose the use
of a additional, two part registry which is external to the
App Store. Similar to the App Store, a part of the registry
is installed directly on the device while the other part of
the registry is located on a central server. We integrate the
registries with the App Store in a transparent manner [33],
by using the App Store link to the App 7. The link is re-
quired to download an App from the App Store if the App
is not available on the mobile device. Both, the local and
the external registry contain data that is required for the in-
vocation such as the (1) binding of Apps to a URI schema,
(2) a link to the App Store, (3) the name of the App and
(4) a App Bundle Identifier. The App Bundle Identifier is
an optional field, since in the current implementation nei-
ther Android nor iOS support the App invocation with the
unique App Bundle Identifier, but we expect future versions
of the operating systems to be able to utilize App Bundle
Identifier to invoke Apps. The functionality of the local reg-
istry is limited in comparison with the external registry. The
local registry acts basically as cache for the remote registry
and offers no functionality to register Apps. This is done with
the remote registry, which provides an user interface

7. ARCHITECTURE AND IMPLEMENTATION

Our proposed architecture addresses some of challenges
we discussed in the previous sections (see Figure 6). In or-
der to cope with the lack of meta data, we propose the use
of a additional, two part registry which is external to the
App Store. Similar to the App Store, a part of the registry
is installed directly on the device while the other part of
the registry is located on a central server. We integrate the
registries with the App Store in a transparent manner [33],
by using the App Store link to the App 7. The link is re-
quired to download an App from the App Store if the App
is not available on the mobile device. Both, the local and
the external registry contain data that is required for the in-
vocation such as the (1) binding of Apps to a URI schema,
(2) a link to the App Store, (3) the name of the App and
(4) a App Bundle Identifier. The App Bundle Identifier is
an optional field, since in the current implementation nei-
ther Android nor iOS support the App invocation with the
unique App Bundle Identifier, but we expect future versions
of the operating systems to be able to utilize App Bundle
Identifier to invoke Apps. The functionality of the local reg-
istry is limited in comparison with the external registry. The
local registry acts basically as cache for the remote registry
and offers no functionality to register Apps. This is done with
the remote registry, which provides an user interface

for the management of App registry data. The remote reg-
istry provides a Rest interface to access Furthermore, our
current implementation, does not include a complex update
process for the local registry other than on demand. A
shown in the figure, our implementation consists of sev-
eral packages that encapsulate the functionality for the QR
barcode reading and the execution of the App composition.
The App Invoker is the central component of our architec-
ture. It is responsible for the binding of the App to an URI
schema and to invoke the App with the data that is being
passed. Therefore, the App invoker queries the local registry
for Apps that are registered with URI schema. In addition,
the App invoker accesses shared memory (e.g., clipboard) to
read data that passed to other Apps during the execution of
the App composition (see Figure 7).

The Data extractor component interprets regular expres-
sions from App compositions to perform basic data manip-
ulations on data from the shared memory. The Context
reader accesses context information that is available on the
device. This includes location, time and user settings like
preferred language. Location is encoded in the geo: URI
schema format, time as unix timestamp and the user lan-
guage setting as encoded in country code top-level domain
format.

The task of the Data Parser is parse data that is encoded
in QR barcodes. The data parser tokenizes the data and
passes it to the App Invoker. Our implementation consists
a set of parsers that are able to handle different kind of data
formats. We’ve implemented a event driven XML parser for
the parsing of the App composition description. Furth-
more, our implementation contains parsers for the following
URI schemata: sms:, mailto:, geo:, http: and tel:. If the
data cannot be parsed by one of the parser, the data is in-
terpreted as plain text.

The QR Code Reader is a wrapper for the open source QR
code reader ZBar 8. Our implementation uses the ZBar
library for extracting the raw data from the scanned QR
barcode. The QR Code Reader component takes screenshots
from the main screen every 200 milliseconds and stops the
scanning process immediately after a QR barcode has been
detected.

8. RELATED WORK
Principles of Service Oriented Architecture are well studied in literature [28]. The move of SOA into the mobile domain focuses mainly on implementation details and address limitations of mobile devices. Jusczak et al present [14] a middleware for service-oriented communication which run on mobile devices. Mobile Web Services Architectures [24] aim at providing alternative representations other than XML-based SOAP and fast communication transport options for mobile Web Services [25] [13]. Other approaches use aspect oriented programming to facilitate the access to Services from mobile devices [26]. The work presented in [38] describes an infrastructure and middleware design which is based on the Jini Surrogate Architecture Specification. Web Service for Devices (http://www.ws4d.org/) aim at porting as much of the SOA stack as possible to embedded and mobile devices and make use of gSOAP implementation [37]. The authors of [22] present a lightweight mobile SOA-based architecture based on J2ME and aim at minimizing the traffic to on from mobile devices. [31] investigates the use of short messages as communication mean between mobile devices to invoke Service asynchronously.

Mobile tagging approaches [12] have been studied in the past in closed environments like museums [8] with the emphasis on human computer interaction and augmented reality [29]. The main focus was to bridge the gap between the virtual and physical world [39] using various technologies like barcodes or RFID [11].

9. CONCLUSION AND FUTURE WORK

We’ve presented an architecture to transfer SOA principles into the mobile domain. In contrast to existing approaches, we took a novel perspective on Apps and App Stores and applied SOA principles. We discussed the limitations of applying SOA principles in the light of our approach and highlighted the challenges. We provided a software architecture and discussed a software prototype that offer features to treat Apps similar to Services in the context of mobile operating systems like Android or iOS. In particular, we showed how existing technologies can be combined in novel manner to create App based Applications using a simple grammar for the construction. We discussed App composition in the context of object hyperlinking.

In future work, we foresee a set of extensions. The current version of the grammar will be extended to support the construction of complex App compositions. Our prototype will be extended in several ways. We are going to implement an Android version and will investigate the feasibility of extending parts of the mobile operating system in order to provide App invocations without requiring to register custom URLs for Apps. And finally, we will elaborate on the aspect of object hyperlinking in App compositions which includes more than a single physical object.

10. REFERENCES


