Fixed-mobile hybrid mashups: experiences and lessons on applying the REST software architecture principles to exposing mobile operator services

Sami Mäkeläinen, Timo Alakoski
sami.makelainen@nsn.com, timo.alakoski@nsn.com
Nokia Siemens Networks
Consulting and Systems Integration
P.O.Box 2, FI-02022 Nokia Siemens Networks, FINLAND

Abstract. Data sources used for creating mashups rarely include anything from the vast potential of the mobile domain. However, many enablers and pieces of useful information already exist for such mashups, both on the device side and on the operator network side. The lack of mashups in this fruitful area stems partly from the difficulties in accessing such data. This paper describes the application of Representational State Transfer (REST) principles to opening up mobile-specific assets in order to enable the creation of compelling “fixed-mobile hybrid” mashups; mashups utilizing information both from existing Internet sources as well as mobile-specific data sources. We present alternative architectures for the deployment of such a system, discuss the ecosystem implications of open APIs and provide lessons learned when implementing a proof-of-concept system with a subset of the designed API and demonstration mashups.

Keywords: REST, mashup, SDP, business models, API

1. Introduction

Mobile operators globally are increasingly facing competition from the Internet; innovative new applications and services are being deployed in manners that either leave the operator completely out of the value chain or reduce mobile operators to bit-pipes, providers of access only. Yet the operators arguably control significant and potentially useful assets that could be beneficial if accessible also by external parties. How could these assets be utilized for differentiation or to gain more revenue?

One of the key challenges that operators face in introducing new services is how to harness the speed and innovative power of the so-called Internet developer community. To solve the time to market challenge, many operators are looking to deploy Service Delivery Platforms (SDP) to help them accelerate the application creation and deployment process [7, 11]. For the most parts, however, SDPs are aimed towards the operators’ own internal application development or to a closed group of partners. This is in stark contrast to the trends in the Internet, where open APIs allow basically anyone easy, open access to the service providers’ data. This open approach is taken by many popular Internet services such as Amazon, Google and Yahoo! In recent years, these APIs have increasingly been designed in a so-called RESTful manner – i.e. following the software architecture design paradigm called Representational State Transfer (REST) [2]. Operators like Vodafone, BT, Orange and Telefonica are also gradually creating more accessible APIs [4, 5, 9, 10, 14] However, simple, open, easy-to-use and fully-featured APIs are still largely absent in the operator domain.

This situation became the motivation to undertake a study to research the possibilities further. The goals of the study were to:

a) Design a generic, simple interface to mobile operators’ main non-voice services and assets based on the REST principles
b) Design the interfaces so as to allow the phone to act as another information source and expose additional data gathered via this channel through the same interface
c) Create proof-of-concept type of mashup applications utilizing both the “operator” REST interfaces as well as open APIs from some Internet services, thus creating “fixed-mobile hybrid mashups”

2. Introduction to REST

REST, short for Representational State Transfer, is a software design architecture originally introduced by Roy Fielding in his Doctoral Thesis in 2000 [1]. Even though REST is not a standard or an official specification, it is a commonly used paradigm for creating open APIs. REST relies on the following key principles [6]:

- State, data and functionality are represented as resources. This means each addressable data entity (like a customer, location or an address) is represented as a resource.
• Every resource is uniquely addressable using a universal syntax for use in hypermedia links; i.e. URLs are used as resource identifiers.

• All resources share a uniform, simple interface for the transfer of state between client and resource, consisting of:
  o A constrained set of well-defined operations
  o A constrained set of content types

• A protocol - typically HTTP – is used which is client-server, stateless, cacheable and layered.

The REST principle can also be described as “a small set of verbs operating on a rich set of nouns”, referring to the restricted set of operations performed on a large set of resources. HTTP is typically used as the protocol on top of which RESTful APIs are implemented as it’s universally supported by the Internet infrastructure. Also, existing HTTP methods can be conveniently mapped to other well-known primitives as can be seen from Table 1 below. In REST, HTTP GET is used for fetching a resource, DELETE is used for deleting it and POST and PUT are used for creating and updating the resource respectively.

<table>
<thead>
<tr>
<th>HTTP</th>
<th>GET</th>
<th>POST</th>
<th>PUT</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRUD</td>
<td>Read</td>
<td>Create</td>
<td>Update</td>
<td>Destroy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SQL</th>
<th>SELECT</th>
<th>INSERT</th>
<th>UPDATE</th>
<th>DELETE</th>
</tr>
</thead>
</table>

Table 1: Mapping of HTTP methods to other well-known primitives

An additional benefit with REST is that in contrast to many traditional Web Services-style APIs, REST-type APIs are quite compact and easily human-readable. This is most clearly illustrated by an example; the following is a fictitious example of a SOAP-request for Singapore’s weather forecast:

```xml
<SOAP-ENV:Envelope
  <SOAP-ENV:Body>
    <ns1:getCityWeather
      xmlns:ns1="Weather">
      <op1 xsi:type="xsd:string">Singapore</op1>
    </ns1:getCityWeather>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Compare this to the REST-style equivalent performed with a clear, easily readable HTTP-request that is also a much more compact representation:

```
GET /restapp/weather/Singapore HTTP/1.1
```

3. RESTful design of exposing mobile-specific resources

To begin the process of designing a RESTful open API for mobile operator assets, one must first determine which functionality and data should be exposed. To do this, a dozen service scenarios focusing on social networking, location and communications were sketched and then analyzed to see what features implementing them would require. Some of these scenarios were later chosen to be implemented as proof-of-concept.

While the exact set of features and functions to expose in real-life deployments will naturally depend on the desires and goals of the deploying operator, the following functionality was deemed as most important to be exposed over the REST-type API:

- Subscriber status information such as presence status, context information and location
- Subscriber profile information such as name, photo, current account balance, address etc
- Access to subscriber voice mails and call logs
- SMS and MMS messaging functionality
- Simple statistics and network information
- Search features; e.g. searching subscribers based on location

Access to all of the functionality would obviously need to be controlled by the means of access control lists and appropriate access rights. With the basic functionalities and information to be available via the API decided, the API itself needs to be designed. Following the abovementioned principles of REST, it needs to be kept in mind that all “objects” need to be exposed as resources and all activities need to be performed via the few available primitives. Therefore, one of the key decisions when designing a RESTful system is to develop a clear and concise URI structure that lends itself to the “small set of verbs operating on a large set of nouns”-model.

Examples of the chosen URI structure, which is mostly self-explanatory, are given below:

```
/users/<UID>
/users/<UID>/status
/users/<UID>/voicemail
/users/<UID>/location
/users/<UID>/status
/users/<UID>/...

/search/location/?...
/stats/users/?...
/info/network/<cellID>/
```

As most operations for such a system are expected to take place on a user or a small set of users, a
structure centered on the user ID, denoted by <UID>, was chosen. Search operations were grouped under /search/ URI prefix and miscellaneous network information (such as the base station location) could be made available under the /info/network URI prefix. As an example, the status information of a user with the user ID “JaneDoe” could be fetched with the HTTP query:

GET /users/JaneDoe/status HTTP/1.1
Host: www.operator.com

4. Architectural implementation options

The RESTful API itself only defines the outward-looking interface of the system; how the back-end is implemented is another matter. Architecturally the simplest solution, shown in Figure 1, is to run the RESTful API and the required server-side functionality as a standalone system. With this scenario, the system can essentially be run by any party as no integration to operator systems is required. Some network features can be implemented by integrating to e.g. 3rd party SMS/MMS service providers.

Figure 1: Standalone system architecture

There are, however, significant disadvantages in using a standalone system. While some features like SMS are relatively easy to implement and even fetching the subscribers’ location is possible either using integrated GPS receivers and/or Cell IDs with a device component, most subscriber information resides on operator systems and would be inaccessible in this scenario. Additionally, subscriber auto provisioning is impossible if pre-existing subscriber records are non-existent or non-accessible.

In order for the operators to get the benefits of the RESTful API, the operator will need to integrate the API elements to “real” back-end services such as the subscriber database, location information server etc.

The Service Delivery Platforms already discussed in the introduction provide one suitable method of tapping into the operator infrastructure without direct integration to all the required network elements. In essence, the RESTful API can be implemented as an application inside the SDP, providing another kind of interface to the outside developers and letting the SDP platform worry about the internal interface details. This architecture option is shown in Figure 2 below.

Yet another option of integrating the RESTful API into the operator systems is to have the RESTful API itself reside on a system which is integrated directly, on an as-needed-basis, to the required elements in the operator infrastructure. This approach, pictured in Figure 3, is most feasible for operators that do not intend to deploy an SDP as this route allows them to bypass the unnecessarily heavy middleware that the SDP would become if only deploying the RESTful API is the goal.

Figure 2: Integration via an SDP

However, disadvantages of this approach include the substantial integration work necessary which, in most cases, cannot be easily re-utilized elsewhere.

Finally, there is an option of the network elements themselves providing an open interface, creating a scenario where the “RESTful API” is distributed to the appropriate elements themselves. This approach is often made impossible by security and network configuration policies that prevent direct external connections from the network elements. Additionally, no unified interface to all data could be presented in this scenario and due to implementation challenges this approach is not viewed as feasible.

In all the above scenarios, there is an optional mobile device element. This indicates a piece of software residing on the consumer handsets that can provide the network-side system with additional information about the client’s location, presence and other state information directly from the user device.

5. Ecosystem implications

It is increasingly believed even by operators themselves that the primary source of service
innovation in the future will not be the traditional operators. This belief is illustrated by JP Rangaswami, the managing director of BT Design, who has said “We watch Google very carefully; we watch Amazon very carefully; we even watch eBay very carefully, because the learning will not come from watching AT&T and Telstra” [10].

Given the potential benefits of the open APIs, it is not surprising that operators like British Telecom, Telefonica, Vodafone and Orange have already begun offering APIs to certain network functionalities [4, 5, 10]. While these APIs typically initially focus on SMS messaging and authentication, more advanced features are also implemented and several more planned. Open APIs from operator environments are also encouraged by industry experts [13]. Yet clearly not all mobile operators are convinced by the opportunities presented by opening up such an API. Such doubt is indeed understandable when one considers the API business case in any detail.

Whether or not one believes in the potential value of the open APIs, the business case itself has yet to be proven. While it is clear that operators control assets that can be extremely useful in creating mashups, it is another question altogether whether subscribers or other parties will be willing or capable of paying for them in one way or another. This uncertainty about direct monetization, however, should not in our view prevent development but is rather an added incentive to recruit the wider Internet developer base to the task.

If one views the first and foremost idea of opening up an API to the Internet developer community as one of fostering innovation, it is a good idea to keep the service initially free if at all possible. Adopting a “free trial”-type of service model where developers are free to try out the system and only need to pay once there is significant usage for their application(s) will inevitably result in the operator incurring some relatively modest initial costs. However, it can be argued that adding complex up-front agreements and charging even relatively small fees raises the barrier of entry too high.

Another important aspect that should be kept in mind is API reachability. There should be as small a number of APIs for the developers to connect to as possible; ideally one standardized or alternatively one centralized API. As there are hundreds of operators in the world, developing a globally usable service would require integration to each of their APIs, which would be tedious at best. This opens up two different kinds of opportunities; first, one for an API broker that could act as an aggregator of operator assets and offer a single, unified interface to the developer world. The second opportunity lies in standardizing the exposed API, a goal where GSMA is aiming with their recently launched initiative called GSM Access API [12].

6. Demonstration system

In order to determine the feasibility of the RESTful API and its usability in service development, a standalone system was created to demonstrate the usage of the RESTful API. When implementing a system designed to access potentially sensitive operator resources, security and access control is always a key issue. The following requirements were set for the system:

- Authentication and authorization of all users and requests must be possible
- Access to all exposed resources must be controlled with a fine-grained access control system
- End users must have complete control on which external parties are allowed to access their information and to what extent such access is allowed.

After analyzing the available options, the OAuth framework was chosen for the proof of concept implementation [3]. Figure 4 shows the call flow of the OAuth authentication/authorization mechanism used. One of the primary benefits gained from using OAuth is that the service providers using the API need not be given the user credentials for accessing the user data. Instead, configurable tokens are used which can be revoked by the user at any time. This ensures that customer data is protected at the operator systems and no unwanted exposure to that data takes place.

![Figure 4: OAuth call flow for initial access](image)

As a general architecture for the implementation, a modified version of the standalone architecture option described in Chapter 4 was selected; the system is in essence standalone, but designed in a way that makes seamless integration to an SDP easy. The overall architecture of the demonstration system is depicted in Figure 5.

The system was modeled so that the REST API component is designed to be run by an operator, whereas a 3rd party “mashup developer” develops and runs the actual mashup code, accessing the customer data through the operator REST API. Customer...
database resides with the operator of the REST API. For managing permissions, a web-based administrative interface was developed for the users. As a return format, the demonstration system supports both XML and JSON formats.

A handset component was also developed for the proof of concept implementation. The handset element periodically submits updates about the device’s location and various presence elements to the network utilizing the RESTful API.

To demonstrate building applications utilizing the RESTful API, a mashup demonstration page as well as a Facebook application called “RESTful location” were developed. The Facebook application can be added to ones profile page and displays the last known location and presence information of the user, overlaid on a map utilizing Yahoo! Maps. The user’s Facebook contacts will then be able to see his/her presence and location information on the Facebook profile page. A screenshot of the application is presented in Figure 6.

Figure 5: Demonstration system high-level architecture

The second demonstration mashup more fully utilizes the RESTful API features. This mashup includes the location of the user superimposed on a map using Google Maps, the user’s presence, currently active phone profile and calendar information of the user, ability to listen to (simulated) voice mails as well as a collection of recent, nearby photos fetched from Flickr. There is also a capability to send the user an SMS, implemented using a 3rd party service provider over the Internet. A screenshot of this mashup is presented in Figure 7.

The easy creation of mashups using the REST API for mobile-specific resources proves that there are no technological barriers to the creation of fixed-mobile hybrid mashups. On the contrary, if present, the utilization of mobile-specific resources through such an API is a simple and straightforward exercise that can be undertaken by any developer familiar with mashup creation in the “fixed” Internet. Even very simple enablers, such the location and presence information, can be used to create compelling “mini-mashups” as exemplified by the “RESTful Location” Facebook application.

Figure 6: RESTful Location Facebook application

Figure 7: Demonstration mashup utilizing several data sources

7. Conclusions

The primary goal of the study was to determine whether it is feasible to expose traditional mobile-operator services via a simple, open, “Internet-style” API in order to create hybrid mashups using data sources both from the mobile domain and the external Internet service providers. This was deemed feasible and it can be stated with confidence that exposing even quite detailed and fine-grained access to operator systems is technologically possible in a simple and open, yet flexible and secure way. In short, designing systems in a RESTful manner is possible even on top of mobile operator environments which have been designed using very different paradigms.

To operators used to dealing with complex interfaces, the promise of a simple, yet powerful API to unleash the creative forces of the Internet developer base may seem dubious. Yet simple APIs do not automatically mean inflexible or somehow inferior functionality when compared with the traditional, rather complex, typically 3GPP-specified APIs prevalent in the operator domain. With some planning, a simple and easy-to-use API can be deployed that still
offers remarkable flexibility and allows 3rd party providers to easily deploy advanced mashups while retaining the capability to use methods and software that the Internet-developers have become accustomed to and without mobile domain-specific special knowledge. Utilizing a possibly pre-existing SDP as a middle-layer over which the RESTful API is built has the advantage of lowering integration effort and being able to utilize existing platforms in a structured manner.

When designing an API that complies with the REST ideology, it is important to take into account some idiosyncrasies that are often present in the mobile operator domain. One of them is security; operators are traditionally very concerned and meticulous when it comes to security. Reconciling the need for simplicity and openness with the operators’ strict security requirements is by no means a simple task but we believe utilizing OAuth-style authorization and TLS encryption for the API access provide the necessary framework for implementing a security architecture that fulfills the needs of both the Internet developers and the operators.

While from a technological point of view the findings are encouraging, the more difficult challenges lie elsewhere; the challenge in integrating the “Internet-world” with the mobile operators’ domain is less about technology and more about reconciling the differences in operational and development philosophies, processes and degrees of openness. For one, mobile operators need to realize that new, innovative services are not necessarily instant financial successes, but rather provide a differentiation opportunity and the potential of becoming a revenue-generating service in the longer run. After all, even some of the most popular Internet services such as Facebook are not yet profitable.

Even the best technical solution by and itself does not allow new, Internet-style innovation to magically happen in the operator domain. The new options and degrees of openness available must therefore not only be properly implemented but also wholeheartedly adopted in the core thinking and then advertised, supported and encouraged by the operator – not immediately limited and charged. Recently there have been some encouraging developments indicating a changing mindset with some operators launching more open APIs and even talking about mashups. Another important aspect to keep in mind is that the developer base would benefit from being able to access as many operators via a single API as possible. This raises the question and potential for either a centralized API and/or a standardized API. It is feasible that the former could be brought to market faster by an “API broker” entity by providing a single, unified interface towards the developers for all participating operators. The latter solution, on the other hand, is being aimed at by the GSMA Access API initiative.

Despite the clear potential that fixed-mobile hybrid mashups present and the increasingly widespread encouraging developments in the operator domain, it is still a long-term goal that fixed-mobile hybrid mashups become as commonplace services as mashups in the Internet are today. It is also uncertain which model will ultimately be the dominant one and if there is more uncertainty to be found from the business models of the open APIs. Despite all the obstacles, it can be stated with a fair degree of confidence that as the potential benefits of opening up mobile-specific assets to the creation of mashups are quite compelling, a model that works for all parties involved will eventually be found.

References