Key Performance Considerations When Designing Mobile Applications Using SOA

A service-oriented architecture can enable IT to consistently create and implement high-performance mobile apps that meet the just-in-time business needs of professionals in life sciences, healthcare and beyond.

Executive Summary
Access to enterprise information is increasing-through mobile channels. Existing information assets are typically being exposed as Web services to enable rapid development of mobile applications. Due to the unique nature of mobile devices, however, performance requirements need to be taken into account.

This white paper discusses key principles that can help IT organizations create mobile apps that can overcome performance challenges. It details the association of a service response object with a UI template, configurable way of contracts maintenance, gateway services as single service invocation and loose coupling. An application built using these principles is discussed with specific reference to how physicians’ needs for information access can be fulfilled in a timely fashion. The paper concludes with the possibilities of application of the suggested principles in other circumstances.

As the Information World Turns
When Confucius said: “They must often change who would be constant in happiness or wisdom” he must have been anticipating today’s world of real-time information ubiquity. Consider how information is disseminated. From print media to the Internet to mobile, ways of reaching consumers are continuously evolving. The first shift from print media to the Internet created all sorts of emotions—cynicism, optimism and guarded hope. That the Internet has permeated every aspect of human existence is hyperbole no longer. Even in common experience, one can notice that newspapers, product brochures and talk shows de facto end with a reference to a Web site for more detailed information.

The force of change was so dramatic that those who failed to move or reluctantly embraced change ended up as anachronisms and consigned to obscurity.

One can see the same parallels experienced in the shift from the Internet to mobile devices. Where earlier the onus was on the user to seek and locate the information on the Web using search engines, today’s models predetermine what users want and create readily consumable information capsules. The ascendancy of such new modes is fueled by the emergence of the mobile Internet. The mobile Internet has empowered on-the-go users with access to the same level of information that is available to a more stationary home Internet user, transcending the limitations of time.
and location. The widespread reach coupled with compactness of the mobile devices has made them attractive and utilitarian. That these devices are usually always on helps ensure immediate distribution of information.

In a pioneering work on mobile information access,1 the authors argue with great foresight that mobile phones will emerge as the preferred and primary platform for information access. The need to be extremely context-aware and understand the intent of the user is predictive to success in the mobile world. An interesting blog from Harvard Business Review2 delineates the innovative, transformational possibilities in health care leveraging mobility trends.

From here, we will share some of our learning in architectural principles that we have successfully applied to realize high-performance mobile applications. We first talk about physicians and their information needs and how the mobile channels are playing a greater role; we then discuss the key architectural principles that can be applied for high performance, and walk through an implementation scenario. We conclude with a discussion of how the architectural principles can be applied in other situations as well.

Physicians and the World of Information: Changing Contours

More than ever before, physicians face the daunting task of staying informed about developments in medicine. The rapid rate of the information explosion, coupled with an increasingly aware and assertive patient community demanding the latest and the best, is further challenging them to stay abreast of new developments.

This information is diverse and it ranges from drug databases to side effects to dosage guidelines to the latest news about the adverse effects of a drug.3 The timeliness of information and the ability to access the same is of paramount importance. Traditional knowledge-gathering mechanisms such as literature review, information from pharmaceuticals companies, subscriptions to information banks and discussions with colleagues are tedious and time-consuming.

While these mechanisms are valid and help build a more comprehensive understanding of medical science developments, oftentimes physicians need to have immediate access to real-time information. Conceivably, such situations occur at the point of care where, as an example, the suitability of a particular drug for the visiting patient may need to be determined.

On the other hand, pharmaceuticals companies realize that multichannel engagement with physicians is essential. They know that enabling physicians to maximize patient interaction is the key to success. Developments in medical science therefore must be converted to a stimulating dialogue with patients. The mutual success of patient-physician interaction typically yields the best possible outcome for both parties. Cognizant of this, pharmaceuticals companies are increasingly looking at creating a means to arm physicians with all necessary information, available on demand, to help determine the most appropriate treatment options for the ultimate benefit of the patient.

Manhattan Research4 suggests that more than 72% of U.S. physicians have smart phones; of that, 80% agree that their devices are essential to their practice. This finding is of crucial importance as it attests to the new means of working for physicians. The adoption of mobile devices by the medical community for professional purposes is set to only expand with time.

Recognizing the prevalence and increased market acceptance of mobile-based applications, in July 2011 the Federal Drug Administration (FDA) released a draft guideline5 on mobile medical applications. The FDA recently approved Mobile MIM, an application that lets physician view for diagnostic purposes CT, MRI and PET images. Similar mobile application developments6 are emerging elsewhere, as well.

With more and more physicians using mobile devices, system architects face the challenge of making information available for effective use in these devices. The devices are constrained in real estate and have much lower processing power compared with desktop and laptop computers. The architectural principles must reflect the understanding of these natural limitations yet not compromise on performance or usability. In the next section we will focus on the architectural principles that are very effective for creating a high-performing, accessible device that delivers a wide array of information to mobile users.
Principles of Performance

The five principles described below are influenced by the following well-established, primitive architectural considerations:

- Clear separation of concerns.
- Grouping of reusable components.
- Drive configuration at all levels.
- Establish contracts between layers and anticipate changes.

**Principle 1: Predefine UI Templates and Associated Response Types**

It is possible to create predefined UI templates for some of the most commonly sought after information. For instance, one can have a list-type display while showing medical news; for media information, one can display a list of images. If the responses from the enterprise information systems are formatted in an appropriate way for predefined methods of rendering (templates), the display in the client devices can be made fast. This also offers an elegant way to manage changes. The idea of predefining the display and contracting with a suitable response structure is at the heart of this architectural principle.

Figure 1 describes some of the UI templates and the contracted response structures.

**Principle 2: Gateway Services – Single Service Invocation**

When the mobile client requests data, it will be ideal if the client is not burdened with having to know the service name and make the necessary request construction. Such an act will consume precious resources in the device and will be detrimental to performance. Gateway services provide a uniform way of constructing the service request irrespective of whatever the client is looking for; be it for medical calculators or label information, the originating request is always made to the gateway services. The URI of the service is always the same and takes the following form: https://<site>/GatewayService.svc. The template name will be appended to the service call and that serves to indicate the kind of information in which the client is interested.

Gateway service also handles another important part of the job: All service invocations from the middle layer are handled by the gateway services. This helps the client insulate itself from the concern of whether the sources are inside or outside the firewall. Figure 2 depicts the core function of gateway services.

Since the handling of all services is centralized to gateway services, this helps in scalability as one can provision necessary hardware depending on the load.

**Principle 3: Configurator – Configurable Contracts Maintenance**

The “Configurator” is a bridging mechanism between the variety of information sources and the client (device). There could be different types of information that a physician may be looking for, so the mode of presentation and the information sources will be different. Much like Web services abstract the underlying information sources and exposes these services in one uniform construct, the Configurator provides the necessary abstrac-

### Interface Options

<table>
<thead>
<tr>
<th>UI Template</th>
<th>Response Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Template</td>
<td>List Response</td>
<td>For applications which are required to display a list of items (in a table view), data will be formatted as list response and will be sent to the client.</td>
</tr>
<tr>
<td>Actions Template</td>
<td>Actions Response</td>
<td>For any action like adding a new app, removing an app or adding an article to my selection, data will be formatted into actions response and the status of the action (success/failure) will be sent to the client.</td>
</tr>
<tr>
<td>Indexed List Template</td>
<td>Category List Response</td>
<td>For any grouping based on category (such as alphabets), data will be formatted in such a manner where items to a particular group will be added under that category and will be sent to the mobile client.</td>
</tr>
<tr>
<td>Media Template</td>
<td>List Response</td>
<td>Media template will display the list of images. The response will contain the image details in a list format and the images will be bound to the media template.</td>
</tr>
</tbody>
</table>

Figure 1
tion and a consistency in structure. At the core of the Configurator is the mapping between the applications, templates, and services. Figure 3 illustrates the core concept.

In implementation, Configurator is an XML component with two major sections:

- **Application:** To act as the bridge between the client and service.
- **Services:** To act as the bridge between the service and its actual implementation.

In the application section of the Configurator, the mapping between the application and the type of template necessary for appropriately rendering the application will be determined. Each application will have a template predetermined. The application section will also map the template to the service. This mapping acts as the bridge between this section (application) and the services section of the Configurator. In the service name section of the Configurator file, the mapping between the service name and the actual service implementation URL is maintained. This mapping will be read at runtime to determine the service to be invoked. Since the services are maintained in a configurable XML file, it provides the dynamism to change the actual service implementations.

For example, to point to different sources for “medical news,” all that the administrator needs to do is to change the URL of the service.

**Principle 4: Loose Coupling at All Levels**

Brittleness in architecture can be removed by having a clear segregation of functionality with a layered approach. This will help isolate the changes in specific areas without impacting the whole system. This will help retain the basic fabric and yet afford easy ways to make changes as required. This principle has been put to very efficient use in our architecture by using the following patterns:

- **Service locator:** Pattern to locate the service and delegates to the next layer with greater context.
- **Service provider:** Possibility to decouple the service implementation from service interface.

**Anatomy of Configurator**
• **End-point channel:** Ability to do service invocations with minimal programming efforts (and thus reduce runtime costs).

**Service locator:** An implementation based on generics, service locator is a design pattern that allows decoupling clients of services (described by a public interface) from the concrete class implementing those services. The constructor of the class registers all the available services in a dictionary. A generic method returns a reference to the correct implementation fetching it from the dictionary. The clients do not know the actual classes implementing the service. They only have to interact with the service locator to obtain an implementation.

**Service provider:** This contains the actual implementation of the service separated from the service interface. Through this decoupling, the service implementation can be evolved without directly impacting service consumers. This can increase the amount of refactoring opportunities and the range of potential consumer programs and corresponding reuse.

**End-point channel:** Service host is configured by creating an end point specifying an address, binding and contract. We can merge an address and a binding with a contract in a channel factory to create a channel. Having an address (https://<site>/GatewayService.svc), a binding (BasicHttpBinding) and a Contract (IListManager), we can create the channel:

```csharp
BasicHttpBinding basicHttpBinding = new BasicHttpBinding();
EndpointAddress url = new EndpointAddress("https://<site>/GatewayService.svc");
IListManager channelClient = new ChannelFactory<IListManager>(basicHttpBinding, url).CreateChannel();
```

Using ChannelFactory, we can generate the proxy based on the interface definition and the configuration. This avoids the need to generate a separate proxy class. This also allows one to code against the service interface.

**Principle 5: Foundation Service Components**

Foundation service components are used across the application to support common functionality. Having this, separately, helps to improve maintainability and reusability for all system-related changes. The main components in the solution are:

- JSON parser.
- Pagination controller.
- Log recorder.

**JSON Parser**

This is primarily used to construct the response. When the enterprise information system returns a list of news headlines, the JSON parser converts the object into JSON format and passes it on to the device. JSON parser is also used to read the incoming request details from a mobile device through a gateway service.

**Pagination Controller**

Pagination information is defined in the configuration file using the number of items per page attribute. Based on the value defined in this attribute, service delivery layer will ensure that only the requested number of items is sent to the device by applying filter logic. For subsequent pages, the device will send the request with the page number so that appropriate items can be retrieved.

**Log Recorder**

This component will be used across all the applications to capture the status of the incoming requests from mobile clients and also to log the status of the requests made by service delivery layer to other external data sources.

**Principles in Action:**

**A Scenario Walkthrough**

Consider an application that bundles all necessary physician information together. Let us further consider a situation where the physician is interested in “medical news.” The request object will be created from the mobile client with the template, app name, device type and device model information as shown in Figure 4. The entry from the figure shows that for medical news application, the UI template to use is the “List” template. The gateway service request that is invoked by the mobile device takes the form:

https://<site>/GatewayService.svc/

The entry from the figure shows that for medical news application, the UI template to use is the “List” template. The gateway service request that is invoked by the mobile device takes the form:

https://<site>/GatewayService.svc/

ListTemplate

This call is handed to the gateway service. The “Gateway Service” will locate the right service using the configurator. From the snippet of configurator shown in Figure 5, the service invocation is:

"http://<SiteName>/SearchService.svc/GetData"
From the gateway services, using the principle of loose coupling, the services are invoked through the service locator and service provider. This enables changing the implementations easily without making invasive changes. When the response is received, using the foundational services of JSON parser, the response is restructured to the format determined by the response object for the UI template.

The overall architecture with the flow and a sample UI is depicted in Figure 6.

- **Mobile device consumes** [https://<site>/Gatewayservice.svc/lListTemplate](https://<site>/Gatewayservice.svc/lListTemplate).
- **Gets the reference of implementation from** service locator.
- **Invokes the actual implementation of the interface.**
- **Retrieves the service name and service value based on the app name and template name.**
- **Consumes external services** [http://<SiteName>/SearchService.svc/GetData](http://<SiteName>/SearchService.svc/GetData).
- **Gets the response in XML or JSON format.**
- **Required data is framed and sent in JSON Format.**
- **Response data to mobile device in JSON format.**

The flow sequence can be summarized as below:

Client -> Gateway Service -> Service Locator -> Service Provider -> Service Provider -> Gateway Services -> Client

As can be evidenced, the architectural approach is highly extensible as there is a clear separation of concern between pure rendering of the UI
and the data elements inside the UI. Consider a scenario where a new information source needs to be added – for example, “medical videos.” Assume further that the Web service to get the medical videos is made available. One can simply associate the medical video information source with an appropriate service, UI template and the response type in the configurator and it will be available in the mobile client without any development efforts.

Conclusion

The architectural principles discussed above offer a great deal of flexibility in adding new information sources. The salient benefits of this architecture are:

- **Device Agnostism**: Decoupled from device and can work with any type of device.
- **Decoupled**: All layers are detached.
- **Configurable**: The display type or service can be changed by invoking simple settings in configuration file.
- **Extensible**: New data sources can be added with ease.
- **Performance**: Delivers high performance as layers are designed to do specific tasks.

Multinational requirements can also be handled with ease using this approach by specifying the language and country in the device request; a service mapping in the configurator can be created for language/country.

The architecture can be used in enabling any type of mobile application from existing enterprise information sources. The separate rendering of UI and data elements within the UI is a powerfully innovative idea that can be put to use in transactional applications as well. Similarly, the level of configuration-based changes that this architecture permits can be used where a need exists to point to different information sources, depending on the level of user role.

We intend to further our work in the area of building extensible mobile enabling layers that can efficiently make available the enterprise information to mobile devices.

**User Interface**
Acknowledgments

The authors express their profound gratitude to Cognizant’s Saikumar Jagannathan, Chief Architect, for his consistent encouragement. Many thoughts expressed herein have been shaped by stimulating discussions with Srivatsan Nagaraja, Cognizant Senior Vice President and head of Life Sciences, North America. We also wish to record our gratitude to Sairamkumar Jeyaraman, Cognizant Vice President, for propelling us to think creatively.

Footnotes

4 http://manhattanresearch.com/.
5 FDA regulations on mobile applications, http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/ucm255978.htm.

About the Authors

Lavanya Hariharan is a Technology Specialist in Cognizant’s Technology Solution’s Life Sciences Practice. Lavanya is passionate about technology and specializes in Microsoft technologies, mobile applications, performance optimizations and estimations. She can be reached at Lavanya.Hariharan@cognizant.com.

Raghuraman Krishnamurty is the Chief Architect in Cognizant Technology Solution’s Life Sciences Practice. He holds a masters degree from Indian Institute of Technology, Mumbai, India. Raghuraman works exclusively for pharmaceuticals clients and specializes in mobility, big data and social media. He is a senior member of the Association of Computing Machinery (ACM) and is based in Chennai, India. He can be reached at Raghuraman.Krishnamurthy2@cognizant.com.

Baskar Senguttuvan is the Principal Architect in Cognizant Technology Solution’s Life Sciences Practice. He holds a Ph.D. Baskar specializes in software architectures, Java and J2EE-based developments and software development methodologies. He can be reached at Baskar.Senguttuvan@cognizant.com.

About Cognizant

Cognizant (NASDAQ: CTSH) is a leading provider of information technology, consulting, and business process outsourcing services, dedicated to helping the world’s leading companies build stronger businesses. Headquartered in Teaneck, New Jersey (U.S.), Cognizant combines a passion for client satisfaction, technology innovation, deep industry and business process expertise, and a global, collaborative workforce that embodies the future of work. With over 50 delivery centers worldwide and approximately 145,200 employees as of June 30, 2012, Cognizant is a member of the NASDAQ-100, the S&P 500, the Forbes Global 2000, and the Fortune 500 and is ranked among the top performing and fastest growing companies in the world. Visit us online at www.cognizant.com or follow us on Twitter: Cognizant.