# Especialización en Sistemas y Servicios Distribuidos

Trabajo Final de Especialización

## Tema

iOS y Android en Entornos Distribuidos

**Tutor**
García Mattío, Mariano

**Alumno**
Mercado, Diego

Fa.M.A.F.

**2011**

# Indice

# Introducción

Dado el auge y continuo crecimiento del desarrollo móvil, es un hecho que son plataformas ideales para el desarrollo distribuido. Ahora bien, no siempre estas brindan absolutamente todas las herramientas necesarias para las mismas: ya sea por límites en la API provista o por la apertura en sí del sistema.
El objetivo de este trabajo implica realizar una comparativa entre dos plataformas líderes actuales del mercado: iOS y Android, en el marco del desarrollo distribuido y evaluar las herramientas nativas con las cuentan cada una y la mención de algunas API *third-party* en caso de su inexistencia.

A continuación haremos una análisis previo sobre:

- Arquitectura
- Memoria
- Seguridad

Luego, nos enfocaremos en el trabajo comparativo en sí, en términos de

- Intérpretes de XML
- Intérpretes de JSON
- Web services (RPC, SOA, REST)
- Servicios Publicación / Subscripción
- QRCode

# Análisis

## *Arquitectura*

## Android

Android ejecuta Java principalmente. Un lenguaje orientado a objetos cuya gran diferencia, respecto de Objetive-C, es que es interpretado. Es decir, el resultado final de compilación es lo que se denomina *bytecode*, datos binarios que luego son convertidos al lenguaje de máquina en tiempo de ejecución.

En particular, Android a diferencia de Java estandar, posee una máquina virtual denominada Dalvik, que transforma las clases compiladas de extensión .class en .dx. A su vez, este compilador permite ejecutar múltiples instancias de máquinas virtuales delegando en el sistema operativo el aislamiento de procesos, memoria e hilos.

La arquitectura del sistema es la siguiente:

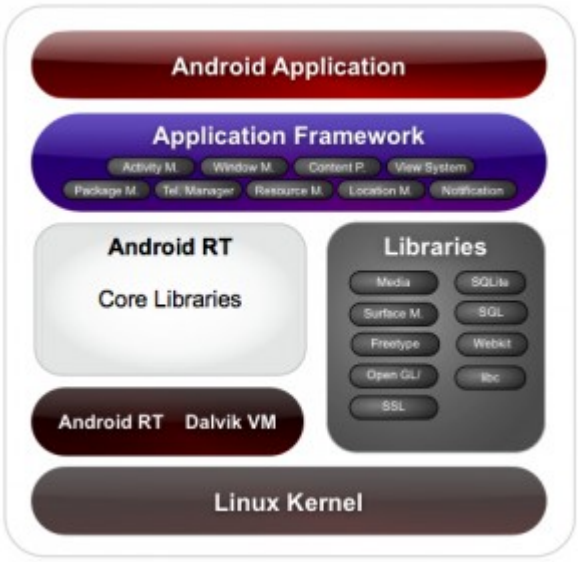



*Figura 1: Arquitectura Android*

1. **Aplicación**: las aplicaciones son escritas en Java. Cabe aclarar, sin embargo, que existe la posibilidad de ejecutar código nativo (C y C++) a través de Android NDK. Este uso se desestima para aplicaciones que no sean críticas en performance, ya que aumenta la complejidad de la implementación y tiene limitaciones en términos de API.
2. **Framework**: consiste en la API en sí escritas en Java
3. **Libraries**: este conjunto de librerías están escritas en C y C++.
4. **Android Runtime (RT) y Dalvik VM**: es el núcleo de las librerías escritas en Java y la máquina virtual Dalvik, que ejecuta código .dex (Dalvik Executable) para traducirlo en lenguaje de máquina
5. **Linux Kernel**: esta capa representa las funciones esenciales del sistema operativo: drivers, acceso al sistema de archivos, red, etc.

## iOS

iOS corre principalmente en Objective-C, un lenguaje orientado a objetos creado como un superconjunto de C, similar a Smalltalk. Compilado para arquitecturas ARM.

La arquitectura del sistema es la siguiente:

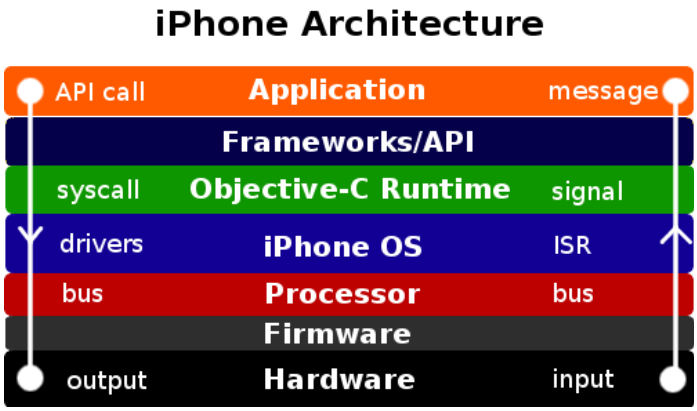



*Figura 2: Arquitectura iOS*

- **Aplicación**: representa la aplicación en sí, compilada a código nativo, y linkeada para correr código Objective C en tiempo de ejecución y librerías escritas en C.
- **Framework/API**: incluye las cabeceras de alto nivel, la mayoría escritas en Objective-C, con algunos linkeos dinámicos en tiempos de ejecución. Abarcan frameworks como Cocoa Touch, llamadas OpenGL de alto nivel, etc.
- **Objective-C Runtime:** librerías de más bajo nivel escritas en Objective-C con linkeos dinámicos, pero que tienen como base funciones escritas en C.
- **iPhone OS**: consiste en el sistema operativo en sí: kernel, drivers, etc.
- **Procesador**: el conjunto de instrucciones de ARM
- **Firmware**: código específico del chip de ARM y de periféricos (como por ej. el giroscopio, la pantalla táctil, etc.)
- **Hardware**: el hardware en sí. Las intrucciones se ubican en las capas superiores.

## *Memoria*

## Android

Java emplea un colector automático de memoria (*Garbage Collector*) que asegura que todas las porciones de memoria obtenidas a través de la creación de objetos que queden sin referencia.

Por ejemplo:

```
private void testMethod1() {
  // inicializamos objeto
  String str = new String();

  // al salir no lo liberamos explícitamente
  // el Garbage Collector lo debería liberar
  // en algún momento
}
```

*Texto 1: Análisis de memoria en Android*

Esto brinda mayor estabilidad como asi también disminuye la performance. Sobre cuando debería liberarlo, la especificación de Java no lo asegura. Depende exclusivamente de la implementación de la máquina virtual de Java y de la implementación del *Garbage Collector*.

A su vez, existe el método System.gc() que indica a la máquina virtual que libere explicitamente la memoria, aunque como la especificación aclara, no garantiza que efectivamente lo haga:

"Indicates to the virtual machine that it would be a good time to run the garbage collector. Note that this is a hint only. There is no guarantee that the garbage collector will actually be run."[1]

En la práctica, los dispositivos móviles poseen una implementación de la máquina virtual más reducida, debido a las limitaciones propias de hardware y resulta una buena práctica llamarla luego de algunas operaciones para forzar una liberación inmediata (por ej.: tratamiento de imágenes) y evitar así un `OutOfMemoryError`.

## iOS

iOS no dispone de un *garbage collector* automático (no así en MacOS). El sistema que propone iOS se refiere a la disciplina de mantener el ciclo de vida de los objetos manteniendo un modelo de contador de referencias.

El modelo funciona de la siguiente forma:

1. Cada vez que se crea un objeto el contador *retainCount* es 1
2. Cuando otros objetos manifiestan la necesidad de tener una referencia a dicho objeto llaman al método *retain* y aumentan este contador en 1
3. Cuando dichos objetos no necesitan de dicha referencia llaman al método *release* y disminuye ese contador en 1
4. Cuando retainCount llega a cero entonces la porcion de memoria dedicada a ese objeto es liberada

1 http://developer.android.com/reference/java/lang/System.html#gc()

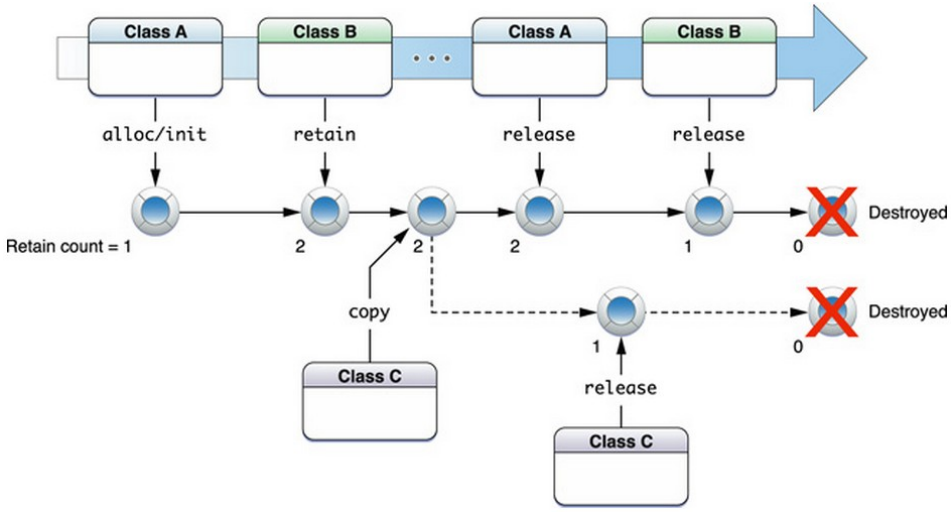



*Figura 3: Modelo de Contador de Referencias - iOS*

Por ejemplo,

```
NSObject* obj1 = [[NSObject alloc] init];
NSLog(@"retainCount (obj1): %i", [obj1 retainCount]);
NSObject* obj2 = [obj1 retain];
NSLog(@"----------------------");
NSLog(@"retainCount (obj1): %i", [obj1 retainCount]);
NSLog(@"retainCount (obj2): %i", [obj2 retainCount]);
[obj1 release];
NSLog(@"----------------------");
NSLog(@"retainCount (obj1): %i", [obj1 retainCount]);
NSLog(@"retainCount (obj2): %i", [obj2 retainCount]);
[obj1 release];
NSLog(@"----------------------");
NSLog(@"(obj1) y (obj2) liberados");
```

*Texto 2: Análisis de memoria en iOS*

Produce la siguiente salida:

```
retainCount (obj1): 1
----------------------
retainCount (obj1): 2
retainCount (obj2): 2
----------------------
retainCount (obj1): 1
retainCount (obj2): 1
----------------------
(obj1) y (obj2) liberados
```

Es extremadamente importante mantener el alcance a nivel de cada clase invocando a lo métodos *retain*/*release*, ya que si el programa intenta emplear un objeto que apunte a una región ya liberada el programa comenzará a realizar un comportamiento no esperado o efectuará una salida anormal (*crash*).

Por otra parte, iOS propone *Autorelease Pools*: básicamente cuando se llama a un objeto a través del método *autorelease,* el objeto marca el mismo para ser liberado dentro de la instancia de NSAutoreleasePool. Cuando esta última se libera entonces todos los objetos marcados para ser

liberados mediante autorelease se liberan.

```
NSObject *obj = [[[NSObject alloc] init] autorelease];
// se liberará cuando se lo indique a la
// instancia de NSAutoreleasePool creada previamente
```

*Texto 3: Análisis de memoria en iOS*

En el último XCode 4.2 con iOS 5, iOS incluyó una funcionalidad opcional denominada *Automatic Reference Counting* (ARC): la misma asegura que un programa no requiere más especificar *release* o *retain*. Lo que provee el compilador LLVM es un proceso de pre-compilación donde los release son agregados automáticamente.

Por ejemplo, si creamos la instancia de un objeto cualquiera:

```
NSObject *obj = [[NSObject alloc] init];
// do some stuff
```

*Texto 4: Análisis de memoria en iOS*

El compilador automáticamente agrega el *release* correspondiente:

```
NSObject *obj = [[NSObject alloc] init];
// do some stuff
[obj release]; // **Added by ARC**
```

*Texto 5: Análisis de memoria en iOS*

Para ello también le da la posibilidad al programador de especificar referencias fuertes o débiles (*strong and week references*), para indicarle al compilador en tiempo de ejecución el ciclo de vida de una instancia.

## *Seguridad*

# Android

### Permisos

Absolutamente todas las aplicaciones en Android no tienen por defecto permisos que puedan impactar sobre otras aplicaciones, el SO o usuarios.

Las mismas se ejecutan en un entorno conocido como "*sandbox*", ya sea que la misma se ejecute como Java o código nativo a través de NDK.

Todos los permisos se conceden en tiempo de compilación, ninguno en tiempo de ejecución para evitar posibles vulnerabilidades. Estos se declaran en el archivo `AndroidManifest.xml` y se le consulta siempre al usuario sobre conceder o no tales permisos antes de la instalación de la aplicación.

### Firma

Las aplicaciones requieren estar firmadas por un certificado generado por el mismo desarrollador. Es importante destacar que no requiere dicho certificado estar avalado por una autoridad certificadora (CA). Android lo emplea para distinguir entre los autores de las distintas aplicaciones.

# iOS

### Firma

Las aplicaciones deben estar firmadas por certificados provistos por *iOS Program Portal*.

Se genera la clave privada localmente a través de una aplicación de escritorio (*Keychain Access*) provista por MacOS.

Estos certificados especifican la aplicación, la configuración para determinados servicios y durante el desarrollo especifica un número limitado de dispositivos en donde se instalaría a través de su identidicado universal (UDID).

### Permisos

En iOS se acceden a los servicios a través de la capa "Core Services". Esta misma implementan servicios basados en Core OS (kernel). Esta capa provee múltiples API que permiten acceder a servicios de seguridad. Por ejemplo, podemos encontrar CFNetwork empleada para comunicaciones seguras.

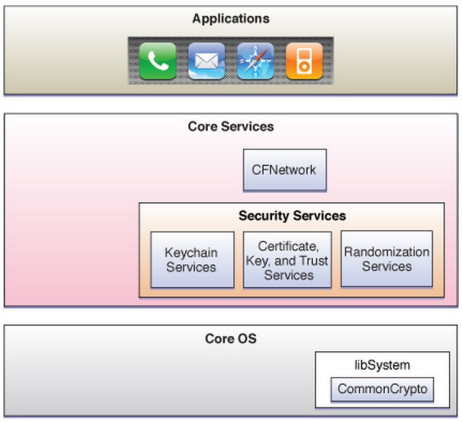



*Figura 4: Arquitectura de Seguridad - iOS*

### Security Server Daemon

Cabe destacar que iOS implementa un proceso en background (daemon), denominado *Security Server*. El mismo no cuenta con una API pública sino que es accedida por las APIs de *Keychain Services*, *Certificate* y *Trust Services*. Estas mismas proveen servicios para: gestionar certificados, agregar certificados a keychain, encriptar/desencriptar datos, verificar firmas y gestionar políticas de confianza (*trust policies*).

### *Interprete de XML*

Para un mismo XML existen dos formas de interpretarlo:

- Almacenar el árbol completamente en memoria y pedir por los elementos luego. Conocido como *Tree-based API* o ***DOM***
- Como un flujo de eventos que se disparan cada vez que se encuentra un elemento. Conocido como *Event-driven API* o ***SAX***

DOM tiene como ventaja la sencillez de interpretación. Además brinda la posibilidad de generar XML a partir de una estructura de objetos. La gran desventaja de este tipo de solución es el empleo de memoria, sumamente limitada en dispositivos móviles.

SAX en cambio, es empleado para XMLs de gran tamaño y minimiza el uso de memoria.

## Android

### *SAX*

Provee en forma nativa las siguientes API:

- `javax.xml.parsers.SAXParser`
    - Este parser exige de un gestor llamado *Handler* que recibe eventos desde la instancia de SAXParser. Este llama a métodos de DefaultHandler para indicar que tag / texto ha encontrado.

```
SAXParserFactory factory = SAXParserFactory.newInstance();
try {
      SAXParser parser = factory.newSAXParser();
      OwnHandler handler = new OwnHandler();
      parser.parse(this.getInputStream(), handler);
} catch (Exception e) {
      throw new RuntimeException(e);
}
```

*Texto 6: Intérprete SAX - SAXParser - Android*

```
public class OwnHanlder extends DefaultHandler{

      @Override
      public void startDocument() throws SAXException {
            super.startDocument();
            // do something....
      }

      @Override
      public void startElement(String uri, String localName, String name,
          Attributes attributes) throws SAXException {
            super.startElement(uri, localName, name,
                                    attributes);
            // do something....
      }

      @Override
      public void endElement(String uri, String localName, String name)
          throws SAXException {
            super.endElement(uri, localName, name);
            // do something....
      }

      @Override
      public void characters(char[] ch, int start, int length)
          throws SAXException {
            super.characters(ch, start, length);
            // do something....
      }
}
```

*Texto 7: Intérprete SAX - Hanlder - Android*

- `org.xmlpull.v1.XmlPullParser`
    - sigue gestionándolo en forma de eventos ya que posee un método que devuelve el siguiente evento encontrado. Su ventaja radica en que la implementación resulta más sencilla. Se emplean identificadores para cada uno: START_TAG, TEXT, END_TAG, END_DOCUMENT

```
XmlPullParserFactory parserCreator =
        XmlPullParserFactory.newInstance();
XmlPullParser parser = parserCreator.newPullParser();
parser.setInput(text.openStream(), null);

int parserEvent = parser.getEventType();
while (parserEvent != XmlPullParser.END_DOCUMENT) {
      switch(parserEvent) {
            case XmlPullParser.START_TAG:
            {
                  String tag = parser.getName();
                  // do something....
                  break;
            }
            case XmlPullParser.TEXT:
            {
                  // do something....
                  break;
            }
      }
      parseEvent = parser.next();
}
```

*Texto 8: Intérprete SAX - XMLPullParser - Android*

### *DOM*

Provee en forma nativa la siguiente API:

- `org.w3c.dom`
    - Este paquete contiene una serie de clases que soporta el estandar W3C correspondiente al nivel 2 (*core level 2*). A continuación dado el siguiente XML demostraremos como recorrerlo:

```
<entry>
      <id>00001112222333444</id>
      <title>Hola Mundo</title>
</entry>
```

*Texto 9: test.xml*

```
InputStream is = this.getResources().openRawResource(R.raw.test);

DocumentBuilderFactory factory = DocumentBuilderFactory
            .newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();
Document dom = builder.parse(is);
Element root = dom.getDocumentElement();
NodeList items = root.getElementsByTagName("entry");

for (int i = 0; i < items.getLength(); i++) {
      Node item = items.item(i);
      NodeList properties = item.getChildNodes();
      // sub-elementos de <entry>
      for (int j = 0; j < properties.getLength(); j++) {
            Node property = properties.item(j);
            if (property.getNodeName().equalsIgnoreCase("title")) {
            // imprimimos el valor correspondiente <title>
                  System.out.println("Title: "
                        + property.getFirstChild().getNodeValue());
            }
      }
}
```

*Texto 10: Intérprete DOM - Android*

Genera como output:

```
Title: Hola Mundo
```

# iOS

## *SAX*

Provee en forma nativa las siguientes API:

- `NSXMLParser` – Foundation.framework

En el siguiente ejemplo, inicializamos el parser:

```
NSXMLParser *addressParser =
      [[NSXMLParser alloc] initWithContentsOfURL:xmlURL];
[addressParser setDelegate:self];
[addressParser setShouldResolveExternalEntities:YES];
success = [addressParser parse]; // return value not used
```

*Texto 11: Intérprete SAX - NSXMLParser - iOS*

Como se observa se está designando como receptor de dichos eventos (*delegate*) a la misma clase, a partir de lo cual se van a llamar a cada uno de los siguientes métodos por cada nuevo evento:

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString
*)elementName namespaceURI:(NSString *)namespaceURI qualifiedName:
(NSString *)qName attributes:(NSDictionary *)attributeDict {
      // do something...
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
      // do something...
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString
*)elementName namespaceURI:(NSString *)namespaceURI qualifiedName:
(NSString *)qName {
      // do something...
}
```

*Texto 12: Intérprete SAX - NSXMLParser - iOS*

### *DOM*

Increiblemente, **Apple no brinda un parser DOM nativo**. Es por ello que debemos recurrir a una implementación *third-party* entre las que podemos encontrar:

- **libxml2**([2]): soporta tanto SAX como DOM. Escrito en C. Licencia MIT
- **TBXML**([3]): soporta DOM, es de solo lectura. No soporta XPath y minimiza el uso de memoria. Licencia MIT
- **TouchXML**([4]): soporta DOM, sólo lectura y soporta XPath. Licencia MIT
- **KissXML**([5]): soporta DOM, permite leer y generar XML. Soporta XPath. Licencia MIT
- **TinyXML**([6]): soporta DOM. Escrito en C. Soporta XPath. Licencia ZLib
- **GDataXML**([7]): soporta DOM desarrollado por Google. Soporta XPath. Licencia Apache

Ray Wenderlich([8]) sugiere la siguiente elección de acuerdo al problema:

- Para lectura de pequeños documentos XML: TouchXML, KissXML y GDataXML debido a su soporte de XPath y si la performance no es tan importante
- Para lectura/escritura de pequeños documentos XML: KissXML o GDataXML, debido a la facilidad de uso y si la performance no es tan importante
- Para lectura de grandes documentos XML: libxml2 o TBXML, ya que aquí la performance y el uso de memoria es crítica

2 http://xmlsoft.org/
3 http://www.tbxml.co.uk/
4 https://github.com/TouchCode/TouchXML
5 http://code.google.com/p/kissxml
6 http://www.grinninglizard.com/tinyxml/
7 http://code.google.com/p/gdata-objectivec-client/source/browse/trunk/Source/XMLSupport/
8 http://www.raywenderlich.com/553/how-to-chose-the-best-xml-parser-for-your-iphone-project

## *Intérpretes de JSON*

## Android

Provee en forma nativa la siguiente API:

- `org.json`

### *Escribir JSON*

Por ejemplo, para escribir nombre, edad y altura:

```
JSONObject object = new JSONObject();
try {
        object.put("nombre", "Diego");
        object.put("edad", new Integer(30));
        object.put("altura", new Double(180.05));
} catch (JSONException e) {
        e.printStackTrace();
}
System.out.println(object);
```

*Text 13: Escribir JSON - Paquete: org.jason - Android*

Generando la siguiente salida:

```
{"altura":180.05,"edad":30,"nombre":"Diego"}
```

### *Interpretar JSON*

Intentaremos leer la salida generada en el anterior ejemplo.

```
try {
      JSONArray jsonArray = new JSONArray(
            "[{\"altura\":180.05,\"edad\":30,\"nombre\":\"Diego\"}]");
      System.out.println("Entradas: " + jsonArray.length());
      for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject jsonObject = jsonArray.getJSONObject(i);
            System.out.println(jsonObject.getString("nombre"));
      }
} catch (Exception e) {
      e.printStackTrace();
}
```

*Text 14: Intrepretar JSON - Paquete: org.jason - Android*

Generando la siguiente salida:

```
Entradas: 1
Diego
```

Observar que la instancia de JSONArray devolvió la existencia de una sola entrada y que al principio y final se agregaron corchetes, de lo contrario hubieramos obtenido una `org.json.JSONException` por la imposibilidad de interpretar dicha cadena de texto.

## iOS

Provee en forma nativa la siguiente API (a partir de iOS 5.0)

- `NSJSONSerialization` – Foundation.framework

Tengamos en cuenta que iOS ha sido liberado recientemente (Octubre 2011). Anteriormente no disponía de tal intérprete en forma nativa.

### *Interpretar JSON*

Consideremos el ejemplo expuesto anteriormente:

```
{"altura":180.05,"edad":30,"nombre":"Diego"}
```

Dado el siguiente código:

```
NSError *error = nil;
NSDictionary *dataDict = [NSJSONSerialization
                            JSONObjectWithData:data
                                   options:NSJSONReadingMutableLeaves
                                         error:&amp;error];
NSString *nombre = [dataDict objectForKey:@"nombre"];
NSNumber *altura = [dataDict objectForKey:@"altura"];
NSLog(@"nombre: %@ - altura: %@", nombre, altura);
```

*Text 15: Intrepretar JSON – NSJSONSerialization - iOS*

Genera la siguiente salida:

```
nombre: Diego – altura: 180.05
```

### *Escribir JSON*

Dado el siguiente código:

```
NSDictionary *data = [NSDictionary
      dictionaryWithObjectsAndKeys:@"180.05", @"altura",@"30",
      @"edad", @"Diego", @"nombre" nil];

NSError *error = nil;
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:data
options:NSJSONWritingPrettyPrinted error:&amp;writeError];

NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];

NSLog(@"%@", jsonString);
```

*Text 16: Escribir JSON – NSJSONSerialization - iOS*

Genera la salida:

```
{"altura":180.05,"edad":30,"nombre":"Diego"}
```

### *Web Services*

Entre los tipos de Web Services más comúnes podemos distinguir los siguientes:

- **Remote Procedure Call (RPC)**: es una función o método que existe en forma distribuida. Se desestima su uso por su alto acoplamiento ya que su implementación está atada a un lenguaje específico
- **Service-Oriented Architecture (SOA)**: la unidad básica de operación es un mensaje, en lugar de ser una operación, por ello se lo conoce como servicios "orientados a mensajes". El WSDL define "el contrato" en lugar de la implementación por ello se lo considera de bajo acoplamiento. Su implementación más común es SOAP
- **Representational State Transfer (REST)**: su foco en lugar de ser las operaciones o mensajes, es la interacción con recursos que gestionan un estado (*stateful resources*). Es por ello que su interfaz se restringe a un protocolo específico. Su implementación más común es sobre el protocolo HTTP empleando las funciones estándar (como PUT, GET, etc..). Por otra parte puede disponer o no de un WSDL (la versión 2.0 ya brinda soporte).

## Android

### *RPC*

No existe soporte de API nativas en forma directa.

Alternativa:

- **JSON-RPC** y **android-json-rpc**: JSON-RPC es una implementación en python y las comunicaciones con la API android-json-rpc se establecen por medio de mensajes JSON ya sea sobre protocolos como HTTP o sockets TCP/IP. La comunicación es relativamente sencilla:

```
JSONRPCClient client = JSONRPCClient.create("http://service/uri");
client.setConnectionTimeout(2000);
client.setSoTimeout(2000);
try
{
      String string = client.callString("mymethod");
      double d = client.callDouble("pow", x, y);
      int i = client.callInt("add", 56, 25); ...
} catch (JSONRPCException e) {
      e.printStackTrace();
}
```

*Texto 17: android-json-rpc*

### *SOA*

No existe soporte de API nativas en forma directa.

Alternativa:

- ***kSoap2-android***[9]: el proyecto *kSoap2* (http://ksoap2.sourceforge.net/) está basado en Java y posee su correlativa implementación en android denominada *kSoap2-android*.

9 http://code.google.com/p/ksoap2-android/

```
final String NAMESPACE = "com.service.ServiceImpl";
final String URL =
      "http://192.168.202.124:9000/AndroidWS/wsdl/ServiceImpl.wsdl";
final String SOAP_ACTION = "ServiceImpl";
final String METHOD_NAME = "message";

SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
SoapSerializationEnvelope envelope =
    new SoapSerializationEnvelope(SoapEnvelope.VER11);

envelope.setOutputSoapObject(request);
HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);
try {
      androidHttpTransport.call(SOAP_ACTION, envelope);
      SoapObject resultsRequestSOAP = (SoapObject) envelope.bodyIn;
} catch (Exception e) {
      e.printStackTrace();
}
```

*Texto 18: kSoap2-android*

### *REST*

El concepto de REST es la comunicación a través de un protocolo ya soportado por la API. En este caso analizamos HTTP.

El paquete `org.apache.http`, incluido en la API nativa de Android, provee clases para establecer una comunicación HTTP..

El siguiente código solicita un respuesta HTTP a partir de `request` (una instancia de `HttpRequest`).

```
HttpClient client = new DefaultHttpClient();
HttpResponse httpResponse = client.execute(request);
responseCode = httpResponse.getStatusLine().getStatusCode();
message = httpResponse.getStatusLine().getReasonPhrase();

HttpEntity entity = httpResponse.getEntity();
if (entity != null) {
      InputStream instream = entity.getContent();
      response = convertStreamToString(instream);

      instream.close();
}
```

*Texto 19: Un conexión simple de HTTP - Android*

`HttpRequest` es una superclase de:

- `org.apache.http.client.methods.HttpPost`
- `org.apache.http.client.methods.HttpGet`
- `org.apache.http.client.methods.HttpDelete`
- `org.apache.http.client.methods.HttpPut`

A partir de cual se pueden generar cualquier pedido HTTP.

## iOS

### *RPC*

No existe soporte de API nativas en forma directa.

Alternativa:

- **xmlrpc**([10]): Escrito en ObjC realiza llamadas RPC a través de XML, como un nombre lo indica. Licencia MIT.

Define un manejador de conexiones (`XMLRPCConnectionManager`) para realizar las llamadas remotas:

```
NSURL *URL = [NSURL URLWithString: @"http://127.0.0.1:8080/"];
XMLRPCRequest *request = [[XMLRPCRequest alloc] initWithURL: URL];
XMLRPCConnectionManager *manager =
      [XMLRPCConnectionManager sharedManager];

[request setMethod: @"Echo.echo" withParameter: @"Hello World!"];
[manager spawnConnectionWithXMLRPCRequest: request delegate: self];
[request release];
```

*Texto 20: xmlrpc – Invocación remota de métodos - iOS*

Y como toda conexión asincróna, posee su propio método para conocer la respuesta a la petición, designado a través de un *delegate* propio:

```
- (void)request: (XMLRPCRequest *)request didReceiveResponse:
(XMLRPCResponse *)response {
      if ([response isFault]) {

      } else {

      }
}
```

*Texto 21: xmlrpc – recepción de respuesta - iOS*

### *SOA*

No existe soporte de API nativas en forma directa.

Alternativa:

- **wsdl2objc**([11]): esta herramienta genera código ObjC a partir del WSDL. Soporte para SOAP. Licencia MIT

El código genera dos importantes clases:

1. **<nombre_del_servicio>Service**: representa al servicio SOAP
2. **<nombre_del_servicio>Binding**: representa el vínculo (*bind*) al servicio. Los métodos de esta clase representan las llamadas asíncronas al servicio
3. **<nombre_del_servicio>Response**: contiene dos arreglos (*headers* y *bodyParts*) que contiene instancias de los elementos autogenerados

Por ejemplo, un request tendría una sintáxis similar a la siguiente:

10 https://github.com/eczarny/xmlrpc
11 http://code.google.com/p/wsdl2objc/

```
MyWebServiceBinding *binding = [MyWebService MyWebServiceBinding];

ns1_MyOperationRequest *request =
      [[ns1_MyOperationRequest new] autorelease];
request.attribute = @"attributeValue";

[binding myOperationUsingParametersAsyncUsingRequest:request];
```

*Texto 22: wsdl2objc – request – iOS*

Donde la respuesta la deberíamos poder distinguir de la siguiente forma:

```
- (void) operation:(FriendsBindingOperation *)operation
completedWithResponse:(FriendsBindingResponse *)response
{
  NSArray *responseHeaders = response.headers;
  NSArray *responseBodyParts = response.bodyParts;

  for(id header in responseHeaders) {
    // do something....
  }

  for(id bodyPart in responseBodyParts) {
    /****
    * SOAP Fault Error
    ****/
    if ([bodyPart isKindOfClass:[SOAPFault class]]) {
      //
      continue;
    }

    if([bodyPart isKindOfClass:
      [types_MyOperationResponseType class]]) {
         types_MyOperationResponseType *body =
(types_getFavoriteColorResponseType*)bodyPart;
         // do something...
         continue;
    }

}
```

*Texto 23: wsdl2objc – response – iOS*

Por instrospección, distinguimos si se trata de la respuesta que estamos esperando. La instancia de SOAPFault indica si existió alguna falla/error de tipo SOAP.

Se sobreentiende que debido a la naturaleza de la herramienta, cada vez que cambia la sintáxis del WSDL, las clases deben volver a generarse.

### *REST*

Como API nativa encontramos el framework CFNetwork. El inconveniente de éste es que resulta tedioso de emplear y se encuentra en muy bajo nivel. En este preciso caso se sugiere emplear la alternativa que mencionamos a continuación: ASIHTTPRequest.

Alternativa:

- **ASIHTTPRequest**([12]): Es un wrapper de CFNetwork API. Podemos hacer todas las operaciones necesarias para una comunicación efectiva con servicios REST (POST, PUT; GET, DELETE). Licencia BSD

Por ejemplo, para realizar un POST de "Hello World", lo podemos hacemos de la siguiente forma:

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request addPostValue:@"Hello World" forKey:@"message"];

[request setCompletionBlock:^{
      NSString *responseString = [request responseString];
      NSLog(@"Response: %@", responseString);
}];

[request setFailedBlock:^{
      NSError *error = [request error];
      NSLog(@"Error: %@", error.localizedDescription);
}];

[request startAsynchronous];
```

*Texto 24: ASIHTTPRequest – POST – iOS*

Para cambiar el anterior request a PUT, lo indicamos sencillamente como sigue:

```
[request setRequestMethod:@"PUT"]
```

*Texto 25: ASIHTTPRequest – PUT – iOS*

12 http://allseeing-i.com/ASIHTTPRequest/

### *Servicios Publicación / Subscripción*

Dependiendo de la comunicación entre un servidor y un cliente, podemos hacer la siguiente distinción:

- **Polling**: Cuando el cliente pide al servidor por datos
- **Pushing**: Cuando el servidor se comunica con el cliente para enviarle datos

Realizar Polling continuamente tiene las siguientes claras desventajas:

- Puede establecer comunicaciones innecesarias cuando no existen datos para traer
- Estresa innecesariamente la red y el ancho de banda
- Consume recursos (cpu, memoria, batería)

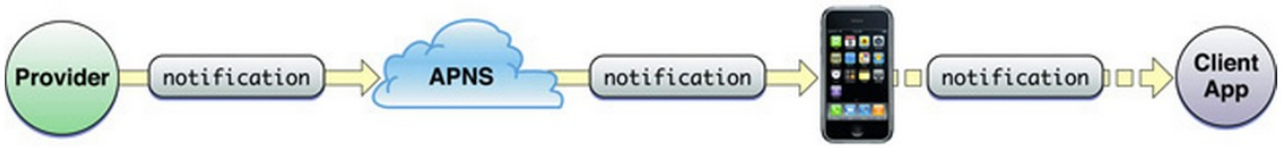
La ventaja de hacer Pushing, ataca necesariamente estos problemas, haciendo mucho más eficiente la comunicación.

## iOS

iOS provee un mecanismo denominado: *Apple Push Notification Services*.

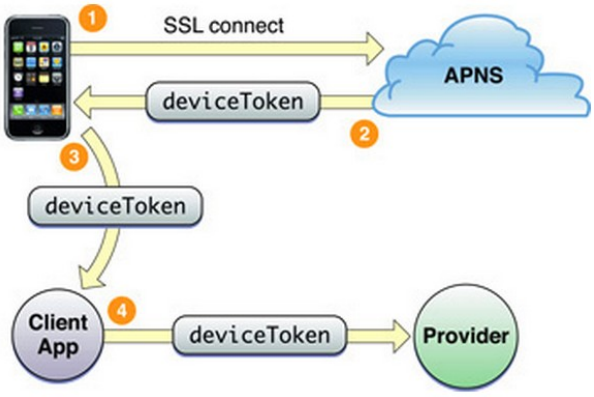
Este permite que cada dispositivo de Apple (iPod, iPhone o iPad) reciba una notificación (en forma de mensaje, alerta o sonido) desde uno de los servidores de la aplicación (*providers*) que se conecta con los servidores de Apple (APNs),

Esta recepción del mensaje se da inclusive aún cuando la aplicación no está ejecutándose.




Cada mensaje que llega a los APNs tiene especificado el proveedor y el device de destino. Este mensaje consta de 2 cuerpos:

- **deviceToken**: contiene un identificador del dispositivo. Esta la obtiene en forma privada, por medio de una conexión SSL, el dispositivo con los servidores de APNS (paso 1 y 2) La aplicación sencillamente implementa un método que recibe dicho *token* (paso 3). Este debe almacenarlo temporalmente e informarlo al propio servidor (paso 4) para una futura comunicación de éste con los servidores de Apple.



- **payload**: es el mensaje en sí. El máximo largo es de 256 bytes. Es un mensaje de tipo JSON que adhiere a la RFC 4627. Básicamente es un diccionario (key-value) que establece una alerta, etiqueta o sonido. A su vez, el mensaje de alerta puede estar localizado. Por ejemplo, el siguiente mensaje muestra una alerta con el mensaje "Tienes emails.", una etiqueta sobre la app con el número 4 y reproduce un sonido.

```
{
    "aps" : {
        "alert" : "Tienes emails.",
        "badge" : 4,
        "sound" : "incoming.aiff"
    },
}
```

*Texto 26: APNS – JSON – iOS*

## Android

A partir de Android 2.2, se creo C2DM que refiere al termino de *Cloud to Device Messaging*.

Al emplear las actuales conexiones de los servicios de Google, requiere que el usuario tenga una sesión iniciada con su cuenta de Google. La aplicación emplea el mecanismo de *Intent* para obtener un ID de registración:

```
// La aplicación usa la API de Intent para obtener el ID de registro
Intent regIntent = new Intent(
        "com.google.android.c2dm.intent.REGISTER");

regIntent.putExtra("app",
        PendingIntent.getBroadcast(this /* your activity */,
            0, new Intent(), 0);

regIntent.putExtra("sender", emailOfSender);

// Iniciamos el servicio
startService(regIntent);
```

*Texto 27: C2DM – Solicitud ID – Android*

```
// La aplicación la obtiene vía Intent
public void onReceive(Context context, Intent intent) {
  String action = intent.getAction();
  if ("com.google.android.c2dm.intent.REGISTRATION".equals(action))
{
    handleRegistration(context, intent);
  }
}
```

*Texto 28: C2DM – Recepción ID – Android*

Una vez que obtiene dicho ID el mismo debe enviarlo a sus propios servidores. Observar la analogía con iOS respecto del *deviceToken*.

Cuando el servidor de la aplicación necesita hacer un *push* de un mensaje dado, realiza un POST HTTP a los servidores de C2DM. Este último redirije el mensaje al dispositivo quien informa de su recepción a través de un *Intent Broadcast* (mecanismo por el cual la aplicación se registró anteriormente para recibir y tiene los permisos adecuados para hacerlo).

Al igual que iOS, no requiere que la aplicación esté ejecutándose.

## *QRCode*

Los código QRCode son códigos de barra matriciales. Estos pueden almacenar cualquier tipo de dato alfanumérico. Se destacan por su gran capacidad de almacenar información y por la recuperación ante errores.

*Figura 5: Ejemplo de QRCode - Texto: FaMAF*

Tanto iOS como Android no soportan la lectura / generación de códigos QR en forma nativa en su API. Como alternativa existe:

- **ZXing**([13]): Permite interpretar y generar códigos de barra 1D/2D. Dispone bibliotecas para la mayoría de las plataformas móviles, incluyendo Android e iOS. Licencia Apache 2.0.

Mostraremos a continuación ejemplos de captura en ambas plataformas:

## Android

En el caso de lectura a través de la cámara de teléfono, iniciamos la lectura de la siguiente forma:

```
IntentIntegrator.initiateScan(YourActivity.this);
```

*Texto 29: QRCode – ZXing – Inicio de Captura – Android*

Obtenemos la respuesta del mismo sobre el contexto (*Activity*) que lo inició:

```
public void onActivityResult(int requestCode, int resultCode,
      Intent intent) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            String contents = intent.getStringExtra("SCAN_RESULT");
            String format =
                   intent.getStringExtra("SCAN_RESULT_FORMAT");
            // Handle successful scan
        } else if (resultCode == RESULT_CANCELED) {
            // Handle cancel
        }
    }
};
```

*Texto 30: QRCode – ZXing – Resultado de Captura – Android*

## iOS

Para realizar exactamente lo mismo que en la implementación de Android

13 http://code.google.com/p/zxing/

```
ZXingWidgetController *widController =
      [[ZXingWidgetController alloc] initWithDelegate:self
            showCancel:YES OneDMode:NO];
QRCodeReader* qrcodeReader = [[QRCodeReader alloc] init];
NSSet *readers = [[NSSet alloc ] initWithObjects:qrcodeReader,nil];
[qrcodeReader release];
widController.readers = readers;
[readers release];

[self presentModalViewController:widController animated:YES];
[widController release];
```

*Texto 31: QRCode – ZXing – Inicio de Captura – iOS*

La obtención de respuesta se realiza por medio de `ZXingDelegate`, en este caso se trata de la misma clase que inició la captura:

```
- (void)zxingController:(ZXingWidgetController*)controller
didScanResult:(NSString *)result
{
      // do something...
}

- (void)zxingControllerDidCancel:(ZXingWidgetController*)controller
{
      // do something...
}
```

*Texto 32: QRCode – ZXing – Resultado de Captura – iOS*

# Conclusión

Solo por mencionar algunas de las API que intervienen en un desarrollo distribuido, hemos podido observar que las carencias de API nativas que abarquen el problema en forma directa. Por ejemplo, el siguiente cuadro ejemplifica dicha situación

| | iOS | Android |
|---|---|---|
| *XML – SAX* | API Nativa | API Nativa |
| *XML – DOM* | **Third-Party** | API Nativa |
| *JSON* | API Nativa | API Nativa |
| *Web Services – RPC* | **Third-Party** | **Third-Party** |
| *Web Services – SOA* | **Third-Party** | **Third-Party** |
| *Web Services – REST* | API Nativa | API Nativa |
| *Push Services* | API Nativa | API Nativa |
| *QRCode* | **Third-Party** | **Third-Party** |

Cabe aclarar que sólo se consideraron algunas alternativas de código abierto (*open-source*) donde a su vez existen infinidad de alternativas privativas.

Esta situación permite generar mercados alternativos que influyen en la calidad y costo de producción de una aplicación dada.

# Bibliografía

## *Principal*


1. “iOS Developer Library”. iOS Dev Center. Apple Inc.
   URL: http://developer.apple.com/library/ios/navigation/
2. “The Developer's Guide”. Android Developers. Google Inc.
   URL: http://developer.android.com/guide/index.html
3. Thomas Erl, Service-Oriented Architecture (SOA): Concepts, Technology and Design. Prentice Hall, 2005.
4. Java Web Services: Up and Running – Martin Kalin. O'Reilly. 2009
5. Android Wireless Application Development 2nd Edition – Shane Conder, Lauren Darcey – 2011 – Ed. Pearson


## *Secundaria*


1. http://es.wikipedia.org/wiki/Dalvik
2. http://en.wikipedia.org/wiki/ARM_architecture
3. http://www.cprogramming.com/compilers.html
4. http://developer.android.com/guide/topics/security/security.html
5. http://developer.apple.com/library/ios/#documentation/General/Conceptual/DevPedia-CocoaCore/MemoryManagement.html
6. http://longweekendmobile.com/2011/09/07/objc-automatic-reference-counting-in-xcode-explained/
7. http://developer.apple.com/library/ios/#documentation/Security/Conceptual/Security_Overview/Architecture/Architecture.html
8. http://www.ibm.com/developerworks/opensource/library/x-android/
9. http://www.cre8ive.kr/blog/2010/04/10/parsing-xml-in-android-dom-method/
10. http://ksoap2.sourceforge.net/
11. http://www.vogella.de/articles/AndroidJSON/article.html
12. http://en.wikipedia.org/wiki/Web_service
13. http://code.google.com/p/ksoap2-android
14. http://stackoverflow.com/questions/8375232/call-a-web-service-from-android-app
15. http://www.codeproject.com/KB/android/webservice-from-android.aspx
16. http://lukencode.com/2010/04/27/calling-web-services-in-android-using-httpclient/
17. http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html
18. http://android-developers.blogspot.com/2010/05/android-cloud-to-device-messaging.html
19. http://www.raywenderlich.com/553/how-to-chose-the-best-xml-parser-for-your-iphone-project
20. http://pragprog.com/magazines/2011-11/inside-ios-
21. http://nathanhjones.com/2011/11/03/getting-started-with-json-in-ios/
22. http://allseeing-i.com/ASIHTTPRequest/
23. http://labs.grupow.com/index.php/2011/03/using-asihttprequest/
24. http://code.google.com/p/zxing/

## Abreviaturas

- **API**: Application Programming Interface
- **ARC**: Automatic Reference Counting
- **C2DM**: Cloud To Device Messaging
- **DOM**: Document Object Model
- **JSON**: JavaScript Object Notation
- **iOS**: Apple's mobile operating system
- **LLVM**: Low Level Virtual Machine
- **NDK**: Native Development Kit
- **REST**: Representational state transfer
- **RPC**: Remote Procedure Call
- **QRCode**: Quick Response code
- **SDK**: Software Development Kit
- **SAX**: Simple Api for XML
- **SO**: Sistema Operativo
- **SOA**: Service-Oriented Architecture
- **SOAP**: Simple Object Access Protocol
- **UDID**: Universal Device Identifier
- **WS**: Web Service
- **WSDL**: Web Service Definition Language
- **XML**: Extensible Markup Language