DSI – EIE
Informática III
Aplicaciones Distribuidas 2011

1. MODELOS CONCEPTUALES

1. El modelo OSI: capas y responsabilidades

Una aplicación distribuida es un conjunto de piezas de software ejecutado en dos o mas dispositivos interconectados. Este conjunto cumple sus objetivos mediante la cooperación de sus distintos componentes, utilizando los enlaces de datos subyacentes para el intercambio de mensajes entre los dispositivos. La figura 1 muestra el modelo arquitectural:

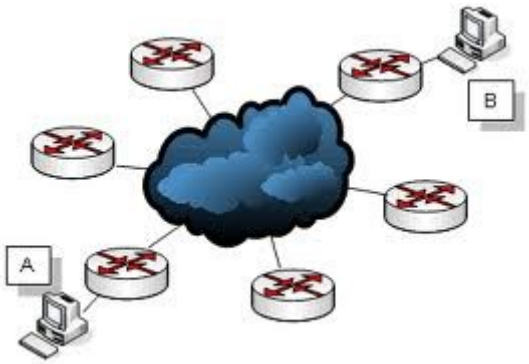

Fig. 1

En esta figura una aplicación que se ejecuta en el dispositivo A coopera con otra que se ejecuta en el dispositivo B. Los segmentos representan enlaces de datos, los dispositivos cilíndricos son enrutadores y la nube representa a la red. Si bien este es el modelo mas general, puede encontrarse múltiples escenarios: desde dos dispositivos vinculados por un cable ("*back to back*", espalda con espalda, ver figura 2) hasta la interconexión sobre Internet.

Fig. 2

El modelo que utilizamos para comprender la interacción es OSI (Open Systems Interconnection) de la ISO, que divide la funcionalidad de la interconexión en siete capas de protocolo:

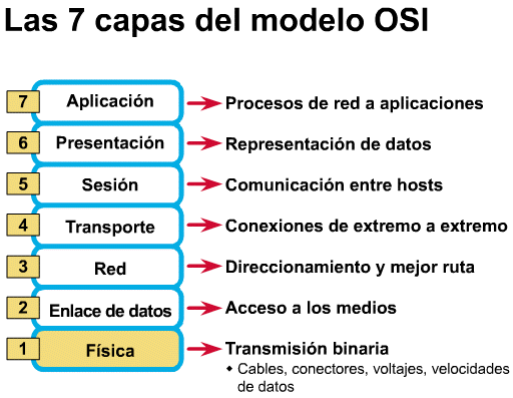

Fig. 3

Utilizando este modelo la comunicación entre aplicaciones residentes en dos nodos de la red será como muestra el diagrama:

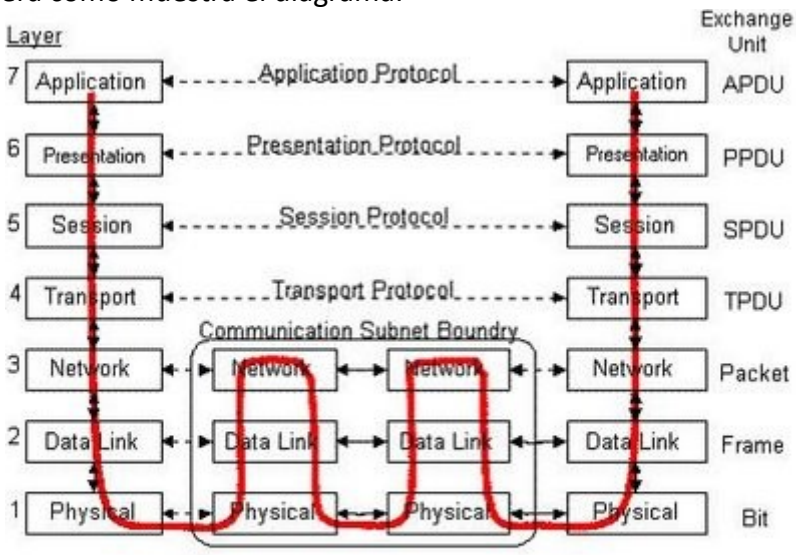

Fig. 4

La rápida adopción del conjunto de protocolos TCP/IP llevó a adoptar el modelo de capas simplificado, que se encuentra en todas las redes TCP/IP como por ejemplo Internet:

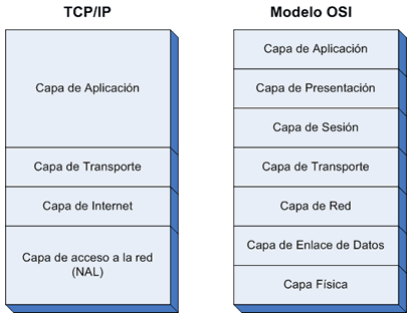

Fig. 5

En este modelo los protocolos de acceso pueden ser: Ethernet, ADSL, MPLS, etc.. En la capa de Internet se utiliza IP (*Internet protocol*) en su versión 4 (actual) o 6 (futura). En la capa de transporte, cuya responsabilidad es transportar mensajes extremo a extremo, es decir, desde el host (nodo) que emite el mensaje hasta el host destinatario, se utilizan los protocolos UDP (*User Datagram Protocol*) y TCP (*Transport Control Protocol*). Para los fines de este capítulo utilizaremos el modelo TCP/IP.

2. Protocolos de aplicación

La "Capa de Aplicación" de la figura 5 es donde residen las aplicaciones distribuidas, como por ejemplo, un servidor de correo electrónico y el programa que utilizamos en una PC, o el servidor Web y el navegador. No está especificada por el modelo TCP/IP, y su diseño y funcionamiento es elección del creador de la aplicación.
Para la interacción de los módulos que corren en distintos nodos de la red debe implementarse un protocolo de aplicación, es decir, un conjunto de reglas para el intercambio e interpretación de los mensajes cursados entre las partes. Tomando un ejemplo de aplicación distribuida como el WWW tendremos:

1. Módulo cliente: el navegador Web (Firefox, Internet Explorer, Safari, opera, etc.)
2. Módulo servidor: el servidor Web (Apache, IIS, etc.)
3. Protocolo de aplicación: HTTP (Hipertext Transfer Protocol)

El protocolo de aplicación se construye utilizando los servicios de la capa de transporte. Esencialmente podemos resumir estos servicios como:

→ Si se usa transporte orientado a conexión (TCP): servicio de envío y recepción de mensajes de longitud arbitraria, con control de errores y garantía de entrega, control de flujo y mecanismos de retransmisión para resolver fallas en los enlaces.

→ Si se usa transporte sin conexión (UDP): envío y recepción de mensajes de longitud fija, sin garantía de entrega ni confirmación de recepción, en el modo "mejor esfuerzo".

3. Middleware: servicios de comunicación para las aplicaciones

La gran difusión de las aplicaciones distribuidas impulsó la creación de herramientas que facilitan su construcción, fundamentalmente orientadas a abstraer la problemática de la comunicación: las fallas en la transmisión de mensajes, la pérdida o alteración de datos en las redes y los retardos originados en la capacidad de los enlaces y nodos intermedios. Si bien hay gran cantidad de herramientas de este tipo, veremos dos en particular muy utilizadas: el modelo de *Sockets* y los *Web Services*.

2. EL MODELO DE *SOCKETS*

1. El stack TCP/IP

El conjunto de protocolos TCP/IP brinda a las aplicaciones que corren en una computadora o dispositivo que lo implemente el servicio de comunicación de datos con distintas modalidades de acuerdo a los requerimientos de cada aplicación, abstrayendo la complejidad de la red subyacente. En efecto, utilizando estos servicios las aplicaciones pueden intercambiar mensajes sin tener directo involucramiento en la gestión de los enlaces, los errores de transmisión o la heterogeneidad de la red.

2. La capa de transporte

La responsabilidad principal de la capa de transporte es recibir mensajes de las aplicaciones y enviarlos al destinatario. En el modelo TCP/IP hay dos modos de transporte: orientado a conexión y sin conexión, soportados por dos protocolos de transporte distintos: TCP y UDP.

TCP garantiza la entrega de mensajes en orden temporal y sin errores en destino, para lo que requiere del establecimiento de una conexión TCP entre el emisor del mensaje y el destinatario. Esta conexión, una vez establecida, brinda un canal bidireccional sobre el cual pueden cursarse mensajes entre ambos extremos. En cualquier momento uno de los extremos puede solicitar la desconexión (llamada *liberación de conexión*) a la contraparte, iniciando entonces un proceso de intercambio de mensajes que finaliza con el canal desconectado e impide transmitir mas mensajes.

UDP, en cambio, brinda el servicio de transporte de datagramas: paquetes de longitud fija hacia un destino en la red, pero no garantiza que efectivamente todos los paquetes lleguen a destino. Es un servicio “no confiable” en el sentido que las aplicaciones que utilicen este transporte no pueden dar por sentado que la contraparte recibe los datagramas o que éstos llegarán en el orden en que fueron enviados. Este modo de operación recibe la calificación de “mejor esfuerzo”. UDP incluye en cada datagrama una secuencia de verificación de trama o *checksum*, que permite detectar alteraciones por errores de transmisión.

3. Canales: extremos, puertos, roles

Cuando se utiliza transporte TCP, una vez establecida la conexión podemos simplificar la red subyacente como indica la figura 6:

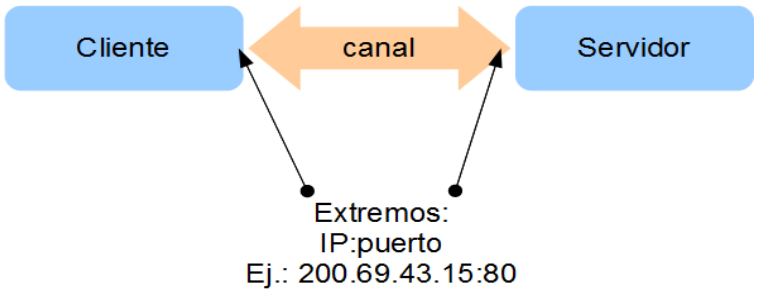

Fig. 6

El canal es una abstracción lógica capaz de transportar mensajes extremo a extremo, donde cada extremo del canal está identificado por una dirección de red, que en las redes IP como Internet es un número de 32 bits[1], y un identificador de puerto, que en las redes IP es un entero de 16 bits (entre 0 y 65535). El puerto es también una abstracción lógica asociada unívocamente a una aplicación que corre en el nodo, y permite entregar los mensajes a ella destinados.
En general el cliente es el nodo que inicia la conexión, mientras que el servidor está a la espera de pedidos de conexión de los clientes. Los servidores usualmente son aplicaciones preparadas para atender múltiples solicitudes de distintos clientes simultáneamente, condicionados por el hardware sobre el que corren.

4. Integración a los lenguajes de programación

Las aplicaciones pueden acceder a los servicios de la capa de transporte utilizando una interfaz (API: *Application Program Interface*) denominada *sockets*, creada en 1983 por la Universidad de California en Berkeley para su sistema operativo BSD Unix, y estandarizada desde entonces para la gran mayoría de sistemas operativos y dispositivos de conectividad en uso. Esta API fue concebida como una librería de lenguaje C, pero hoy puede utilizarse desde muchos lenguajes y plataformas de programación diferentes. Por ejemplo, pueden usarse sockets desde Java, C, C++, Perl, Python, COBOL, .NET, etc.
La gran aceptación de este modelo ha permitido estandarizar su soporte en dispositivos embebidos, tales como microcontroladores, móviles (celulares) y de diversa aplicación (cámaras de video, sensores, etc.).
Si bien depende del lenguaje de programación a utilizar, la interface de *sockets* se presenta como un conjunto de funciones que pueden invocarse desde un programa de aplicación (ej.: C) o un conjunto de clases que permiten implementar objetos (ej.: Java, C++)
Los sockets pertenecen a dos clases de acuerdo al transporte que utilizan. Cuando se implementan sobre TCP se denominan "*stream sockets*" y cuando utilizan UDP "*datagram sockets*".

5. Sockets en lenguaje C

El lenguaje C fue el primero en soportar el uso de *sockets*: la librería original de Berkeley está escrita en C. Esta API está basada en un conjunto de funciones y estructuras de datos que permiten el establecimiento de conexión y su posterior liberación, el envío y recepción de datos y diversas funciones de configuración y control.
La referencia bibliográfica 1 tiene un ejemplo de aplicación en Unix.
El flujo de llamadas a función se muestra en las figuras siguientes:

1 Si se utiliza IP versión 4. En el caso de IPV6 son 128 bits

Stream Sockets – Transporte TCP:

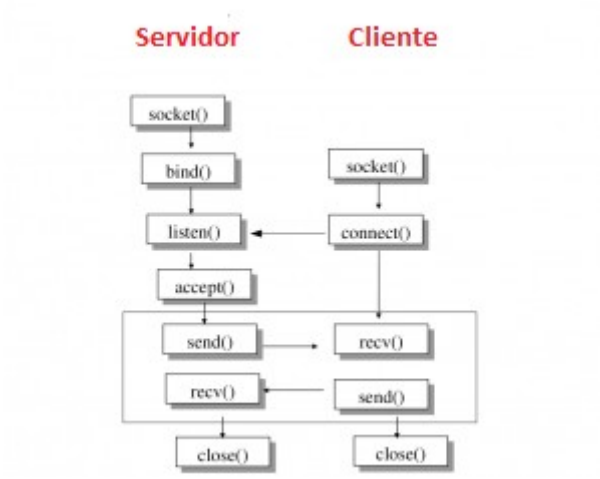

Fig. 7

Datagram sockets – Transporte UDP

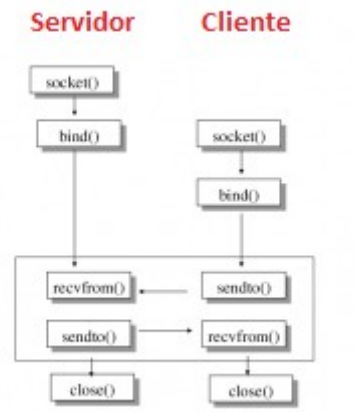

Fig. 8

En el primer caso el servidor puede mantener múltiples conexiones TCP con distintos clientes, utilizando recursos de concurrencia (*threads*) si están disponibles en la plataforma. Cada vez que se recibe una solicitud de conexión, la llamada a función accept() provee un manejador a un nuevo socket sobre el cual se cursarán los mensajes desde y hacia el cliente, en forma independiente del resto.
En el caso de Datagram Sockets no existe conexión, y en cada llamada a función sendTo() o recvFrom() está involucrada la dirección y el puerto de la contraparte.
Ejemplos y detalle de las funciones pueden encontrarse en las referencias bibliográficas 1 y 2.

6. Sockets en Java

El lenguaje Java ofrece un API orientada a objetos para la utilización de sockets, soportando las dos modalidades básicas (streams y datagrams) en el paquete de clases java.net. Allí encontramos clases que implementan los conceptos ya analizados:

1. Socket: lado cliente de una conexión TCP
2. ServerSocket: lado servidor de múltiples conexiones TCP
3. DatagramSocket: socket sobre transporte UDP.

La referencia bibliográfica 3 documenta este tema, y la referencia completa, con ejemplos de programación puede encontrarse en la referencia 4.

3. EL MODELO DE SERVICIOS WEB

1. Arquitectura

1. Definición: Los servicios web (WS) son sistemas de software identificados por un URI (*Uniform Resource Identificator*), cuyas interfaces públicas están definidas y descriptas en XML (*eXtensible Markup Language*). Su definición puede ser "descubierta" por otros sistemas de software, los cuales pueden interactuar con el servicio web en la modalidad prescripta por su definición utilizando mensajes XML transportados por los protocolos de Internet. Por la neutralidad implícita en su definición, el modelo WS permite la interacción entre aplicaciones que corren en computadoras distintas, independientemente del lenguaje de programación o la plataforma que cada una emplea.

2. XML: es un estándar de representación de datos independiente de la plataforma e interpretable por distintas computadores y lenguajes de programación. Está basado en texto y utiliza etiquetas para identificar los datos:

```
<sensor>
<nombre_sensor>ST001</nombre_sensor>
<rango_sensor>-30:+200</rango_sensor>
<tipo_sensor>Temperatura</tipo_sensor>
</sensor>
```

   Lo encerrado entre signos "<" y ">" se denomina "markup" y el resto "contenido"
   Los elementos esenciales de XML son:
   Etiquetas: el contenido del markup. Hay etiquetas de apertura y cierre
   Elementos: Lo comprendido entre una etiqueta de apertura (en el ejemplo <rango_sensor>) y una de cierre (</rango-sensor>), es decir, la cadena de caracteres "-30:+200".
   Atributos: Construcción de markup que contiene pares "nombre/valor", como por ejemplo <img src="diagrama.jpg"> el elemento img tiene un atributo src con valor "diagrama.jpg"

3. URI: (*Uniform Resource Identifier*) es una cadena de caracteres que identifica a un nombre o recurso en Internet. Esta identificación permite la interacción con las representaciones del recurso y los protocolos utilizados para acceder a él. Está descripto y normalizado por la RFC3305.

   Un ejemplo de URI:
   http://dsi.fceia.unr.edu.ar/query.pdf

4. Modelo de interacción
   En la arquitectura WS hay tres roles principales:
   Clientes (*Service Consumers*): son aplicaciones que requieren servicio, como por ejemplo, acceder a un dispositivo remoto o hacer una búsqueda en una base de

datos
Servidores (*Service Providers*): son aplicaciones diseñadas de acuerdo al estándar WS para brindar servicios
Broker de servicios (*Directory*): aplicaciones que permiten a un cliente encontrar el servidor adecuado para la tarea que debe realizar

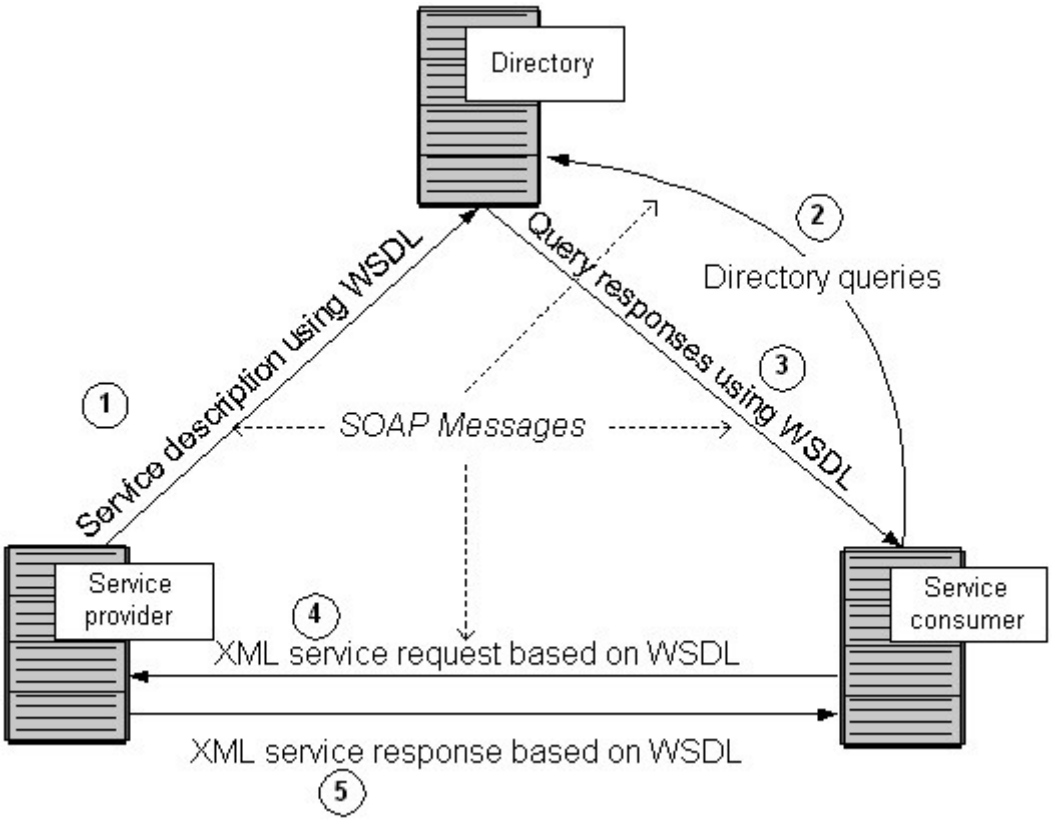

Fig. 9

En la figura los números indican el ordenamiento temporal de los mensajes.
Una vez creada la aplicación que proveerá servicios vía WS en el Service Provider, debe registrarse en el Directorio utilizando el lenguaje de especificación WDSL (*Web Services Description Language*), representado en XML (1).
Cuando un cliente requiere un servicio, envía un mensaje XML (2) sobre Internet al Directorio preguntando por un servidor asociado. Si el Directorio tiene registrado un servicio acorde, envía al cliente una respuesta XML (3) en el formato especificado por WSDL, indicando las características, interface y ubicación del servidor.
Una vez recibido este mensaje el cliente pasa a solicitar el servicio que necesita al servidor (4) también en XML indicando la operación solicitada y los eventuales parámetros requeridos para su ejecución. Al recibir este mensaje, el servidor lo procesa, elabora la respuesta y la empaqueta en un mensaje WSDL (5) que envía al cliente.

5. Aplicación
   Los WS se han diseminado profusamente en Internet, por sus características de interoperabilidad y relativa facilidad de implementación. En la actualidad se han

convertido en un estándar de interconexión de aplicaciones distribuidas y podemos encontrar implementaciones de WS para automatización y control, por ejemplo en el modelo OPC-UA (www.opcfoundation.org) o en el proyecto Web Services for Devices (www.ws4d.org) , especialmente dirigido a la integración de dispositivos embebidos a la arquitectura orientada a servicios.
Para el caso particular de integración de dispositivos en el modelo de WS se utiliza el estándar Device Profiles for Web Services (DPWS). Para una ampliación del tema ver http://ws4d.e-technik.uni-rostock.de/dpws/

6. Arquitectura Orientada a Servicios
   SOA por sus siglas en inglés es la concepción del diseño de aplicaciones integrando servicios de distintos "proveedores" que funcionan bajo el estándar de Web Services. Utilizando esta concepción, una aplicación está compuesta por múltiples invocaciones de servicios web, distribuidos sobre una red y orquestados por un proceso que los integra.

2. Transporte

Los WS están concebidos para su aplicación en redes IP, paradigmáticamente Internet. Para el transporte de los mensajes XML descriptos en el modelo de interacción se utilizan distintos protocolos, mayoritariamente SOAP (Simple Object Access Protocol). SOAP es un protocolo diseñado específicamente para el intercambio de mensajes XML y la utilización de llamadas a procedimientos remotos (RPC) entre sistemas heterogéneos, que puede utilizar servicios de transporte UDP, multicast y HTTP (TCP). Los mensajes SOAP tienen dos partes fundamentales: el encabezamiento y el cuerpo, como se indica:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
      <!— Aquí va el contenido del encabezamiento —>
      ...
  </SOAP:Header>
  <SOAP:Body>
      <!— Aquí va el contenido del cuerpo —>
      ...
  </SOAP:Body>
</SOAP:Envelope>
```

Un mensaje SOAP puede especificar el propósito mediante la etiqueta SOAPAction, así como parámetros para el servicio demandado y el tipo de respuesta esperado.
El protocolo SOAP está implementado en gran número de plataformas de cómputo, notoriamente en aquellas orientadas al desarrollo de aplicaciones distribuidas como .Net y Java. Para los dispositivos embebidos existen varias implementaciones comerciales y de código abierto, tales como gSOAP (http://www.cs.fsu.edu/~engelen/soap.html) para C++, JMEDS para Java y Web Services on Devices para las plataformas Microsoft (http://msdn.microsoft.com/en-us/library/aa826001%28v=VS.85%29.aspx)

3. Descripción de WS

Como se dijo, después de crear un WS debe darse una descripción de sus características, el formato de invocación del servicio y el de las respuestas. Esta descripción se hace utilizando WSDL (*Web Services Description Language*), una representación XML de las interfaces del WS y cómo pueden localizarlo los clientes. Abajo se muestra un ejemplo WSDL de un servicio de reserva de pasajes de avión:

```
<message name="GetFlightInfoInput">
   <part name="airlineName" type="xsd:string"/>
   <part name="flightNumber" type="xsd:int"/>
</message>
<message name="GetFlightInfoOutput">
   <part name="flightInfo" type="fixsd:FlightInfoType"/>
   </message>
<message name="CheckInInput">
   <part name="body" element="eticketxsd:Ticket"/>
</message>

<portType name="AirportServicePortType">
   <operation name="GetFlightInfo">
      <input message="tns:GetFlightInfoInput"/>
      <output message="tns:GetFlightInfoOutput"/>
   </operation>
   <operation name="CheckIn">
       <input message="tns:CheckInInput"/>
   </operation>
</portType>
```

4. Registro y localización de WS

Como se dijo en la sección Arquitectura, un componente esencial del modelo de WS es que los clientes puedan buscar y localizar servidores que entreguen los WS que ellos requieren. Dos mecanismos implementados son UDDI y WS-Discovery, en UDDI existe un repositorio centralizado llamado "WS-Directory" que recibe de los servidores el WSDL que describe a los servicios que se prestan; este directorio es contactado por los clientes que requieren servicio tal como se explicó mas arriba en Arquitectura. En 2009 la organización de estandarización OASIS adoptó otro mecanismo que prescinde del repositorio centralizado, especialmente útil en aplicaciones de automatización, dispositivos embebidos o móviles, donde el cliente que busca un servicio emite requerimientos en modo multicast a su red local describiendo el servicio que necesita. Si hay en ese ámbito uno o mas servidores apropiados, estos responden en forma directa al cliente identificándose, quien a continuación puede solicitar el servicio requerido en forma directa. Esta interacción está descripta en http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.pdf. WS-Discovery presenta un modelo mas simplificado que el de UDDI, y está en proceso de adopción por compañías tecnológicas líderes.

5. Coordinación

Tal como fue definida, al Arquitectura Orientada a Servicios (SOA) concibe a las aplicaciones como la composición de servicios brindados por distintos componentes (servidores) localizados en distintos nodos de la red. Si bien este enfoque ofrece un gran potencial para el diseño de aplicaciones complejas, debe tenerse en cuenta que una aplicación así ensamblada lleva implícita una nueva problemática, la de la sincronización. En efecto, las invocaciones a WS ubicados en distintos nodos de la red pueden ser respondidas por los servidores en orden diferente al de los requerimientos, por lo que es necesario establecer modelos de sincronización. Estos modelos reciben la denominación de "orquestación" de WS, por la similitud con la tarea de un director de orquesta que coordina el trabajo de varios músicos.

Existen numerosos estándares propuestos para la coordinación de aplicaciones basadas en WS, en particular el lenguaje BPEL (Bussiness Process Execution Language) es ampliamente utilizado para la composición de aplicaciones de sistemas.

Por otra parte, la especificación para el tratamiento de eventos en WS, WS-Eventing, permite a un cliente ser notificado de sucesos de su interés por la contraparte.

6. Seguridad

La naturaleza intrínsecamente insegura de las redes y los sistemas distribuidos tiene gran relevancia en el modelo de WS, dado que los mensajes XML entre servidor y cliente deben atravesar enlaces de datos en los que pueden sufrir alteraciones maliciosas. Incluso un atacante puede "disfrazar" sus intenciones ofreciéndose como un WS "normal", o suplantando uno ya existente. Para enfrentar este problema se recurre a técnicas de encriptado de mensajes (por ejemplo, usando transporte https en lugar de http) y certificación de identidades.

En particular la organización de estándares abiertos Oasis-Open ha publicado la especificación del protocolo Web Services Security (WS-Security, WSS). El protocolo contiene especificaciones sobre cómo debe garantizarse la integridad y seguridad en los mensajes utilizados por los Servicios Web. El protocolo WSS incluye detalles en el uso de SAML y Kerberos, y formatos de certificado tales como X.509. Para mayo detalle puede verse la especificación de OASis-Open en http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

7. Integración a los lenguajes de programación

La gran mayoría de los lenguajes de programación en uso en la actualidad soporta el diseño e implementación de clientes y servidores WS. Así por ejemplo las herramientas de desarrollo integrado (IDE) dan la posibilidad de transformar automáticamente una clase Java en una colección de Web Services, donde cada método público es un servicio distinto. Lo propio vale para otras plataformas de desarrollo de software, como .Net.

Para el caso de lenguajes procedurales se dispone de librerías de funciones, siendo el caso mas notorio C.

8. El caso de los dispositivos embebidos y móviles

Como se dijo antes, la gran adopción de los WS ha impulsado su utilización como técnica de interface en dispositivos con menor poder de cómputo, notoriamente los dispositivos basados en microcontroladores o los teléfonos móviles. En estos dispositivos hay una serie de restricciones inexistentes en computadoras de mayor porte. La capacidad de procesamiento, la memoria RAM disponible, el tipo de enlace de datos, etc. hacen que las implementaciones de WS en estas plataformas estén muy ajustadas en cuanto a la demanda de recursos. A pesar de estas restricciones la aparición de modelos como los ya citados DPWS y WS4D permiten una utilización intensiva de los WS en dispositivos. Los sistemas operativos modernos ya incorporan protocolos tales como WS-Discovery que les permiten, por ejemplo, detectar automáticamente la presencia de dispositivo como una impresora o un sensor en la red local. Un ejemplo de aplicación puede encontrarse en la referencia bibliográfica 9. La figura muestra el conjunto de tecnologías implicadas:

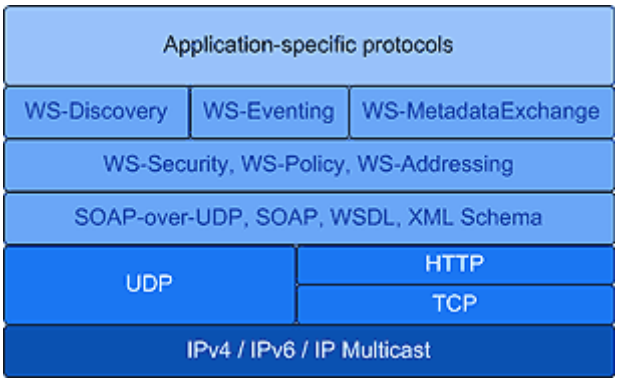

Fig. 10

4. Bibliografía y referencias

1. *An Introduction to sockets programming,* *http://is0dvil.wordpress.com/2010/01/18/introduction-to-socket-programming/*

2. *Overview of Computer Networks, Norman Matloff,* *http://heather.cs.ucdavis.edu/~matloff/Networks/Intro/NetIntro.pdf*

3. *Socket Programming, Nikhil Shetty, 2006* *http://www-inst.eecs.berkeley.edu/~ee122/sp06/LectureNotes/Socket%20Programming.pdf*

4. *The Java Tutorial: custom networking* *http://download.oracle.com/javase/tutorial/networking/index.html*

5. *Web Services explained*, http://www.service-architecture.com/web-services/articles/web_services_explained.html

6. *Web Services Architecture Requirements*, http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/#id2604831

7. *Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI*, Francisco Curbera, Matthew Duftler, Rania Khalaf,William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. IBM T.J.Watson Research Center, IEEE Computer Magazine, Mayo 2002.

8. *Embedded Systems Integration Using Web Services*, Machado, Siqueira, Mittmann, Vieira e Vieira, Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06) (Disponible en la página web de la cátedra)

9. *Web Services for Devices*, http://ws4d.e-technik.uni-rostock.de/dpws/