

SOA aplicado a REST

Profesor: Alejandro Sirvent Llamas

Introducción a los Servicios Web.
Introducción a REST.
SOAP vs REST.
Metodología SOA aplicada a REST.
Integración de Rest en BPEL



Introducción a los Servicios Web

Introducción a los Servicios Web

- Tecnología orientada a la interoperabilidad de aplicaciones
 - Paradigma tecnológico B2B
- Regido por W3C y OASIS
- Múltiples definiciones
 - W3C
 - Sistema software diseñado para soportar la interoperabilidad máquina a máquina sobre la red descrito a través de una interfaz en un formato legible y procesable por una máquina.
 - Aplicación software identificada por una URI, cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML. Los Servicios Web hacen posible la interacción entre "agentes" software (aplicaciones) **"utilizando mensajes XML"** intercambiados mediante protocolos de Internet.
- La Web como base para la construcción de aplicaciones distribuidas
 - Basados en tecnologías surgidas alrededor de la Web

Introducción a los Servicios Web

- Tipos
 - SOAP (Simple Object Access Protocol)
 - Modelo orientado a exponer lógica de negocio
 - Independiente del protocolo de transporte
 - Transparencia en el mensaje de respuesta
 - CORBA, DCOM, etc.
 - Llevar características de entornos distribuidos a la Web
 - Estilos
 - RPC
 - Mensaje
 - Rest (Representational State Transfer)
 - “REST” es un paradigma arquitectónico y “RESTful” describo el uso de dicho paradigma.

Introducción a los Servicios Web

- Tipos
 - SOAP (Simple Object Access Protocol)
 - Rest (Representational State Transfer)
 - Estilo arquitectónico.
 - Basado en tecnología existente (HTTP y XML/JSON).
 - Ligero.
 - Parte de los conceptos de la Web.
 - Obtener información expuesta en la Web.
 - Centrado en interactuar con **recursos**.

```
<root>
  <item>
    <one>foo</one>
    <two>bar</two>
    <three>
      <four>foobar</four>
    </three>
  </item>
</root>
```

XML To JSON

```
{
  one: 'foo',
  two: 'bar',
  three: 'foobar'
}
```

Introducción a los Servicios Web

- Relación con SOA
 - Modelo tecnológico que ha propiciado su éxito (Marks and Bell, 2006)
 - SOA (Erl, 2005)
 - Modelo conceptual = SOA
 - Modelo tecnológico = servicios Web
- Principios de SOA
 - Entidades autónomas y autocontenidas
 - Reusables
 - Duraderos
 - Contratos bien definidos
 - Interoperables
 - Con capacidad de composición
 - Funcionalidad de alto nivel de abstracción
 - Funcionalidades de grano grueso (coarse grained)
 - Abstracción de la lógica subyacente
 - Acoplamiento débil
 - Con capacidad de descubrimiento, publicación y localización
 - Alineados con el negocio
 - Sin estado

- Usos
 - Rest
 - Servicio Web sin estado (http)
 - Sistema Web con caché si no hay datos dinámicos
 - Descripción servicio (OpenApi, Raml...)
 - **Necesidad de rendimiento**
 - Facilidad de integración con Web existentes
 - SOAP
 - Servicio Web sin estado, pero con posibilidad mantener estado de las **operaciones**.
 - Contrato formal (WSDL)
 - **La arquitectura debe soportar requerimientos complejos no funcionales**
 - Seguridad, calidad de servicio, transacciones, direccionamiento
 - Proceso asíncronos

Introducción a REST

Introducción a REST

- REST es un **estilo de arquitectura de software** para sistemas hipermedias distribuidos tales como la Web.
- El término **fué introducido** por Roy Fielding en 2000, (uno de los principales autores de la especificación de HTTP).
- REST **no es un estándar**, ya que es tan solo un estilo de arquitectura.
- REST se refiere estrictamente a una **colección de principios** para el diseño de arquitecturas en red.
 - Estos principios resumen como los recursos son definidos y diseccionados.
- Aunque REST no es un estándar, está basado en estándares:
 - HTTP
 - URL
 - Representación de los recursos: XML/HTML/GIF/JPEG/...
 - Tipos MIME: text/xml, text/html, ...

Introducción a REST

- REST **describe a cualquier interfaz** que transmita datos específicos de un dominio sobre HTTP sin una capa adicional, como hace SOAP.
 - Es posible diseñar un sistema software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding **sin utilizar HTTP** o sin interactuar con la Web.
 - Así como también es posible diseñar una simple interfaz XML+HTTP que **no sigue los principios REST**, y en cambio seguir un modelo RPC.
- La **motivación** de REST es la de capturar las características de la Web que la han hecho tan exitosa.

Introducción a REST

- Principios de REST:
 - Escalabilidad de la interacción con los componentes.
 - La Web ha crecido exponencialmente sin degradar su rendimiento.
 - Gran variedad de clientes que pueden acceder a través de la Web:
 - Estaciones de trabajo.
 - Sistemas industriales.
 - Dispositivos móviles,...
 - Generalidad de interfaces.
 - Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial.
 - Puesta en funcionamiento independiente.
 - Compatibilidad con componentes intermedios.

Introducción a REST

- Principios de REST:
 - Escalabilidad de la interacción con los componentes.
 - Generalidad de interfaces.
 - Puesta en funcionamiento independiente.
 - Los clientes y servidores pueden ser puestos en funcionamiento durante años.
 - Por tanto, los servidores antiguos **deben ser capaces de entenderse** con clientes actuales y viceversa.
 - **Diseñar un protocolo** que permita este tipo de características resulta **muy complicado**.
 - **HTTP permite la extensibilidad** mediante el uso de las cabeceras, a través de las URIs, a través de la habilidad para crear nuevos métodos y tipos de contenido.
 - Compatibilidad con componentes intermedios.

Introducción a REST

- Principios de REST:
 - Escalabilidad de la interacción con los componentes.
 - Generalidad de interfaces.
 - Puesta en funcionamiento independiente.
 - Compatibilidad con componentes intermedios.
 - Los más populares intermediarios son varios tipos de **proxys** para Web.
 - Algunos de ellos, las **caches**, se utilizan para mejorar el rendimiento.
 - Otros permiten reforzar las políticas de seguridad: **firewalls**.
 - Otro tipo importante de intermediarios, **gateway**, permiten encapsular sistemas no propiamente Web.
 - Por tanto, la compatibilidad con intermediarios nos permite reducir la latencia de interacción, reforzar la seguridad y encapsular otros sistemas.

Introducción a REST

- REST logra satisfacer estos objetivos aplicando varias restricciones:
 - Identificación de recursos y manipulación de ellos a través de representaciones.
 - Esto se consigue mediante el uso de **URIs** .
 - Los recursos son los **objetos lógicos** a los que se le envían mensajes.
 - Los recursos **no** pueden ser **directamente** accedidos o modificados.
 - Se trabaja con representaciones de ellos.
 - Internamente el estado del recurso puede ser cualquier cosa desde una BD relacional a un fichero de texto.
 - Mensajes auto descriptivos.
 - Hipermedia como un mecanismo del estado de la aplicación

Introducción a REST

- REST logra satisfacer estos objetivos aplicando varias restricciones:
 - Identificación de recursos y manipulación de ellos a través de representaciones.
 - Mensajes auto descriptivos.
 - REST dicta que los mensajes HTTP deberían ser tan **descriptivos** como sea posible.
 - Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario.
 - HTTP es un protocolo sin estado y cuando se utiliza adecuadamente, es posible **interpretar cada mensaje sin ningún conocimiento** de los mensajes precedentes.
 - Hipermedia como un mecanismo del estado de la aplicación

Introducción a REST

- REST logra satisfacer estos objetivos aplicando varias restricciones:
 - Identificación de recursos y manipulación de ellos a través de representaciones.
 - Mensajes auto descriptivos.
 - Hipermedia como un mecanismo del estado de la aplicación
 - El estado actual de una aplicación Web debería ser **capturada** en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor.
 - **El servidor conoce el estado de sus recursos**, aunque no intenta seguirle la pista a las sesiones individuales de los clientes.
 - El navegador sabe como **navegar de recurso a recurso**, recogiendo información que necesita o cambiar el estado.

Introducción a REST

- ¿Cómo sería un ejemplo basado en REST?
 - Basado en operaciones CRUD.
 - Diseñadas para operar con datos atómicos dentro del contexto de una transacción con la base de datos.
 - Los métodos HTTP más importantes son PUT, GET, POST y DELETE.
 - Suelen ser comparados con las operaciones asociadas a la tecnología de **BD**, operaciones **CRUD**: CREATE, READ, UPDATE, DELETE.

Acción	HTTP	SQL	Copy&Paste	Unix Shell
Create	PUT	Insert	Pegar	>
Read	GET	Select	Copiar	<
Update	POST	Update	Pegar después	>>
Delete	DELETE	Delete	Cortar	Del/rm

Introducción a REST

- Crear una interfaz basada en Rest
 - Asociar recursos a URIs
 - Ejemplo:
 - Una URI por cada empleado de una organización (employee)
 - Una URI para Listado de todos los empleados (AllEmployee)
 - Formato del mensaje
 - No se puede acceder directamente al recurso, hay que obtener una representación.
 - Esta representación puede ser un documento HTML, XML, una imagen,... dependiendo de la situación.
 - Métodos soportados en cada URI
 - Códigos de estado que serán devueltos

Introducción a REST

- Crear una interfaz basada en Rest

```
<employee xmlns='HTTP://example.org/my-example-ns/'>
  <name>Full name goes here.</name>
  <title>Persons title goes here.</title>
  <phone-number>Phone number goes here.</phone-number>
</employee>
```

```
<employee-list xmlns='HTTP://example.org/my-example-ns/'>
  <employee-ref href="URI of the first employee"/>
  Full name of the first employee goes here.</employee>
  <employee-ref href="URI of employee #2"/>Full name</employee>
  .
  .
  <employee-ref href="URI of employee #N"/>Full name</employee>
</employee-list>
```

Introducción a REST

- Crear una interfaz basada en Rest
 - Asociar recursos a URIs
 - Formato del mensaje
 - Métodos soportados en cada URI
 - El acceso se va realizar por medio de las URIs.
 - El acceso se puede hacer de muchas formas:
 - Recibiendo una representación del recurso (GET o HEAD).
 - Añadiendo o modificando una representación (POST o PUT)
 - Eliminando algunas o todas las representaciones (DELETE).
 - Códigos de estado que serán devueltos
 - No solo es necesario conocer que tipo de representación va a ser devuelta, también es necesario enumerar los **códigos de estado HTTP** típicos que podrían ser **devueltos**.

HTTP	CRUD	Descripción
POST	CREATE	Crear un nuevo recurso
GET	RETRIEVE	Obtener la representación de un recurso
PUT	UPDATE	Actualizar un recurso
DELETE	DELETE	Eliminar un recurso

Recurso	Método	Representación
Employee	GET	Formato del empleado
Employee	PUT	Formato del empleado
Employee	DELETE	-
All Employees	GET	Formato de la lista de empleados
All Employees	POST	Formato de empleado

Recurso	Método	Representación	Códigos de estado
Employee	GET	Formato del empleado	200, 301, 410
Employee	PUT	Formato del empleado	200, 301, 400, 410
Employee	DELETE	-	200, 204
All Employees	GET	Formato de la lista de empleados	200, 301
All Employees	POST	Formato de empleado	201, 400

Introducción a REST

- ¿Cómo diseñar un servicio Web basado en REST?
 - Las **pautas** a seguir en el diseño de servicios Web basados en REST son:
 1. Identificar todas las entidades conceptuales que se desean exponer como servicio.
 2. Crear una URL para cada recurso. Los recursos deberían ser nombres no verbos (acciones).
 - Por ejemplo no utilizar esto:
 - » `http://www.service.com/entities/getEntity?id=001`
 - **getEntity** es un verbo. Mejor utilizar el estilo REST, un nombre:
 - » `http://www.service.com/entities/001`

Introducción a REST

- ¿Cómo diseñar un servicio Web basado en REST?
 3. **Categorizar los recursos** teniendo en cuenta si los clientes pueden obtener una representación del recurso o si pueden modificarlo:
 - Para el primero, debemos hacer los recursos accesibles utilizando un HTTP **GET**.
 - Para el último, debemos hacer los recursos accesibles mediante HTTP **POST, PUT y/o DELETE**
 4. Todos los recursos accesibles mediante GET **no** deberían tener **efectos secundarios**.
 1. Los recursos deberían devolver la representación del recurso.
 2. Invocar al recurso no debería ser el resultado de modificarlo.

- ¿Cómo diseñar un servicio Web basado en REST?
 5. **Especificar el formato** de los datos de respuesta mediante un esquema (DTD, W3C Schema, ...).
 - Para los servicios que requieran un POST o un PUT es aconsejable también proporcionar un esquema para especificar el formato de la respuesta.
 6. **Describir** como nuestro servicio ha de ser invocado, mediante una **API**.



SOAP vs REST

SOAP vs REST

- **SOAP → demasiado complicado.**
- **REST → Sencillez.**
- El problema principal surge del **propósito inicial de SOAP.**
 - Esta tecnología fue originalmente pensada para ser una versión, sobre Internet, de DCOM o CORBA.
 - Estas tecnologías basadas en RPC, son más adecuada para entornos aislados, donde se conoce perfectamente el entorno.

- Cuando el número de usuarios es **muy grande** es necesario emplear una estrategia diferente.
- Se necesita proponer un mecanismo explícito para la interoperabilidad de los sistemas que no poseen la misma API.
- Se intenta tomar como **modelo a la Web**.

SOAP vs REST

- La principal ventaja de **REST** recae en la **potencial escalabilidad** de este tipo de sistema, así como el acceso con **escaso consumo de recursos** a sus operaciones debido al limitado número de operaciones y el esquema de direccionamiento unificado.

Características de REST y SOAP

	REST	SOAP
Características	<ul style="list-style-type: none">• Las operaciones se definen en los mensajes.• Una dirección única para cada instancia del proceso.• Cada objeto soporta las operaciones estándares definidas.	<ul style="list-style-type: none">• Las operaciones son definidas como puertos WSDL.• Dirección única para todas las operaciones.• Múltiple instancias del proceso comparten la misma operación.

SOAP vs REST

	REST	SOAP
Ventajas Declaradas	<ul style="list-style-type: none">• Bajo consumo de recursos.• Las instancias del proceso son creadas explícitamente.• El cliente no necesita información de enrutamiento a partir de la URI inicial.• Los clientes pueden tener una interfaz “listener” (escuchadora) genérica para las notificaciones.• Generalmente fácil de construir y adoptar.	<ul style="list-style-type: none">• Fácil (generalmente) de utilizar.• Las operaciones complejas pueden ser escondidas detrás de una fachada.• Envolver APIs existentes es sencillo• Incrementa la privacidad.

SOAP vs REST

	REST	SOAP
Posibles Desventajas	<ul style="list-style-type: none">• Gran número de objetos.• Manejar el espacio de nombres (URIs) puede ser engorroso.• “La descripción sintáctica/semántica muy informal (orientada al usuario)”.	<ul style="list-style-type: none">• Los clientes necesitan saber las operaciones y su semántica antes del uso.• Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones.• Las instancias del proceso son creadas implícitamente.

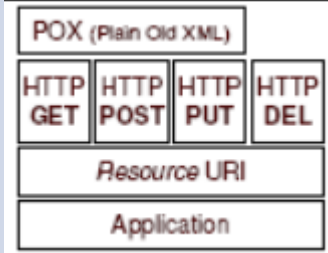
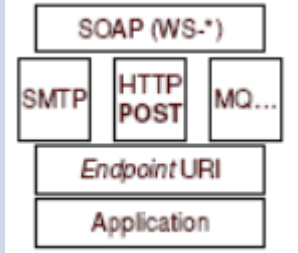
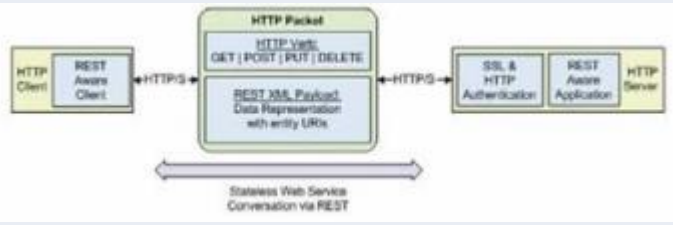
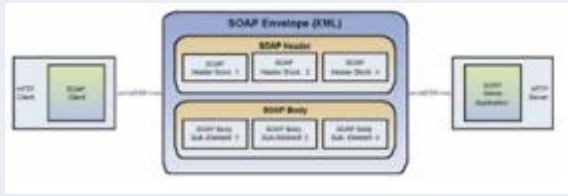
SOAP vs REST

- Principales diferencias entre REST y SOAP

Tecnología	REST	SOAP
	“La Web es el universo de la información accesible globalmente” (Tim Berners Lee)	“La Web es el transporte universal de mensajes”
	ESB BPEL For Rest	Flujo de eventos orquestados. (BPEL)
	Mecanismo consistente de nombrado de recursos (URI).	Falta de un mecanismo de nombrado.
	Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia	Se centra en el diseño de aplicaciones distribuidas.

SOAP vs REST

- Principales diferencias entre REST y SOAP

	REST	SOAP
Protocolo		
	 <p>Stateless Web Service Conversation via REST</p>	
	XML autodescriptivo	Tipado fuerte, XML Schema
	HTTP	Independiente del transporte.
	HTTP es un protocolo de aplicación	HTTP es un protocolo de transporte
	Síncrono	Síncrono y Asíncrono.

SOAP vs REST

- Principales diferencias entre REST y SOAP

	REST	SOAP
Descripción del servicio	Diseño de APIs mediante: OpenApi, blueprint, Raml	WSDL
	Se pueden construir automáticamente stubs (clientes) por medio de OpenApi, RAML...	Se pueden construir automáticamente stubs (clientes) por medio del WSDL.
	No es necesario el tipado fuerte, si ambos lados están de acuerdo con el contenido.	Tipado fuerte.
	OpenApi, RAML,...	WSDL 2.0.

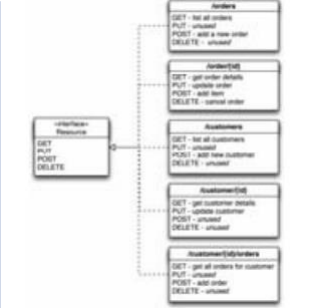
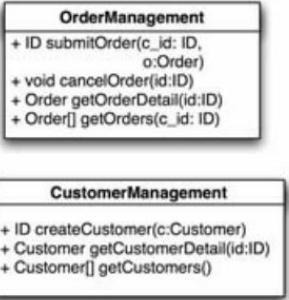
SOAP vs REST

- Principales diferencias entre REST y SOAP

	REST	SOAP
Gestión del estado	El servidor no tiene estado (stateless).	El servidor puede mantener el estado de la conversación (operaciones).
	Los recursos contienen datos y enlaces representando transiciones a estados válidos.	Los mensajes solo contienen datos.
	Los clientes mantienen el estado siguiendo los enlaces.	Los clientes mantienen el estado suponiendo el estado del servicio
	Técnicas para añadir sesiones: Cookies	Técnicas para añadir sesiones: Cabecera de sesión (no estándar)
Seguridad	HTTPS.	WS-Security.
	Implementado desde hace muchos años	Conjunto de especificaciones.
	Comunicación punto a punto segura.	Comunicación origen a destino segura.

SOAP vs REST

- Principales diferencias entre REST y SOAP

	REST	SOAP
Metodología de diseño		
	Identificar recursos a ser expuestos como servicios.	Listar las operaciones del servicio en el documento WSDL.
	Definir URLs para direccionarlos.	Definir un modelo de datos para el contenido de los mensajes.
	Distinguir los recursos de solo lectura (GET) de los modificables (POST,PUT,DELETE).	Elegir un protocolo de transporte apropiado y definir las correspondientes políticas QoS, de seguridad y transaccional
	Implementar e implantar el servidor Web.	Implementar e implantar el contenedor del servicio Web.

SOAP vs REST

- ¿Aplicaremos REST siempre? ...**NO**, ¿Cuándo?
 - Asincronía:
 - SOAP puede utilizarse de una manera asíncrona por medio del uso de un transporte asíncrono.
 - Seguridad
 - “REST no dispone de mecanismos tan completos de seguridad como es el caso de la especificación WS-Security para SOAP”
 - En REST es posible utilizar las características de seguridad implícitas de HTTP:
 - HTTPS y el sistema de autorización y autenticación de HTTP.
 - Descripción del servicio
 - Familiaridad

SOAP vs REST

- ¿Aplicaremos REST siempre? ...**NO**, ¿Cuándo?
 - Asincronía:
 - Seguridad
 - Descripción del servicio
 - SOAP tiene WSDL.
 - Existen varios framework para realizar “design-first” , para servicios REST.
 - Familiaridad
 - REST requiere repensar el problema en términos de manipulación de recursos direccionables en vez de llamadas a métodos.

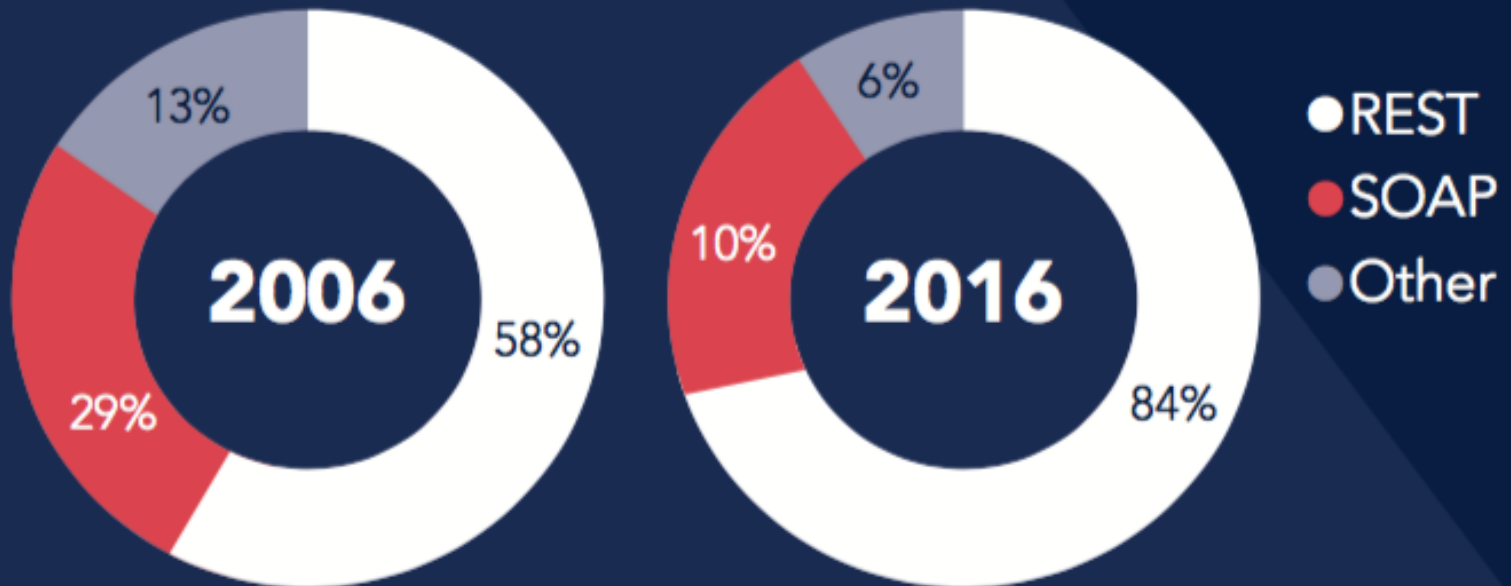
- ¿Dónde es útil REST?
 - El servicio Web **no tiene estado**.
 - Una buena comprobación de esto consistiría en considerar si la interacción puede sobrevivir a un reinicio del servidor.
 - Una infraestructura de **caching** puede mejorar el rendimiento.
 - Si los datos que el servicio Web pueden ser cacheados, entonces se puede incrementar el rendimiento.

- ¿Dónde es útil REST?
 - El **ancho de banda** es importante y necesita ser **limitado**.
 - REST es particularmente útil en dispositivos con escasos recursos como tablets o teléfonos móviles, donde la sobrecarga de las cabeceras y capas adicionales de los elementos SOAP debe ser restringida.
 - La distribución de Servicios Web o la agregación con sitios Web existentes puede ser **fácilmente desarrollada** mediante REST.

SOAP vs REST

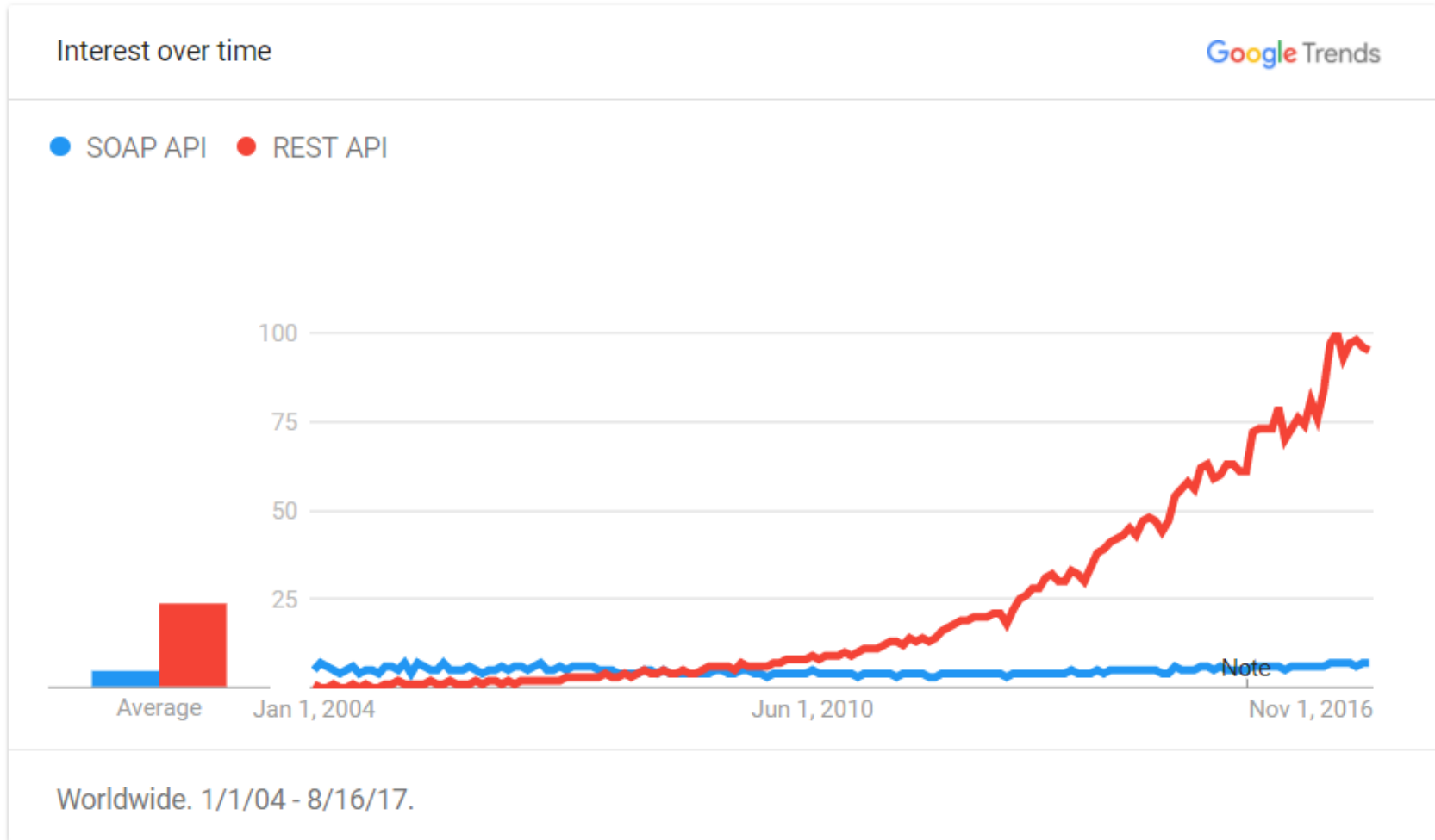
- ¿Dónde es útil SOAP?
 - Se establece un **contrato formal** para la descripción de la interfaz que el servicio ofrece.
 - El lenguaje de Descripción de Servicios Web (WSDL), como ya sabemos, permite describir con detalles el servicio Web.
 - La arquitectura debe abordar **requerimientos complejos no funcionales**.
 - La mayoría de aplicaciones del mundo real se comportan por encima de las operaciones CRUD y requieren mantener información contextual y el estado conversacional, incluyendo **transacciones, seguridad, direccionamiento**.
 - Con la aproximación REST, abordar este tipo de arquitecturas resulta más complicado.
 - La arquitectura necesita manejar procesado **asíncrono**.

- **Conclusión:**
 - Amazon posee ambos estilos de uso de sus servicios Web. Pero el 85% de sus clientes prefieren la interfaz REST.
 - A pesar de la promoción que las empresas han invertido para ensalzar a SOAP, parece que es evidente que los desarrolladores prefieren, en algunos casos, la aproximación más sencilla: REST.
 - El éxito de REST es evidente.
 - En el mundo de la tecnología **no siempre acaba triunfando la tecnología mejor:**
 - VHS vs BetaMax.



Distribution of API protocols and styles, based on directory of APIs listed at ProgrammableWeb, May 2016.

Introducción a REST



Metodología SOA aplicada REST

- **Contrato Uniforme**

- Cuando se utiliza un servicio REST, un recurso es solicitado por el consumidor del servicio.
- Esta **petición** se produce a través de un **contrato uniforme** que será **estandarizado** en toda la **empresa**.
- Este nivel de estandarización reduce los requisitos de acoplamiento entre el consumidor y los servicios.

Metodología SOA aplicada REST

resource identifier syntax
(e.g. URL)

methods
(e.g. HTTP)

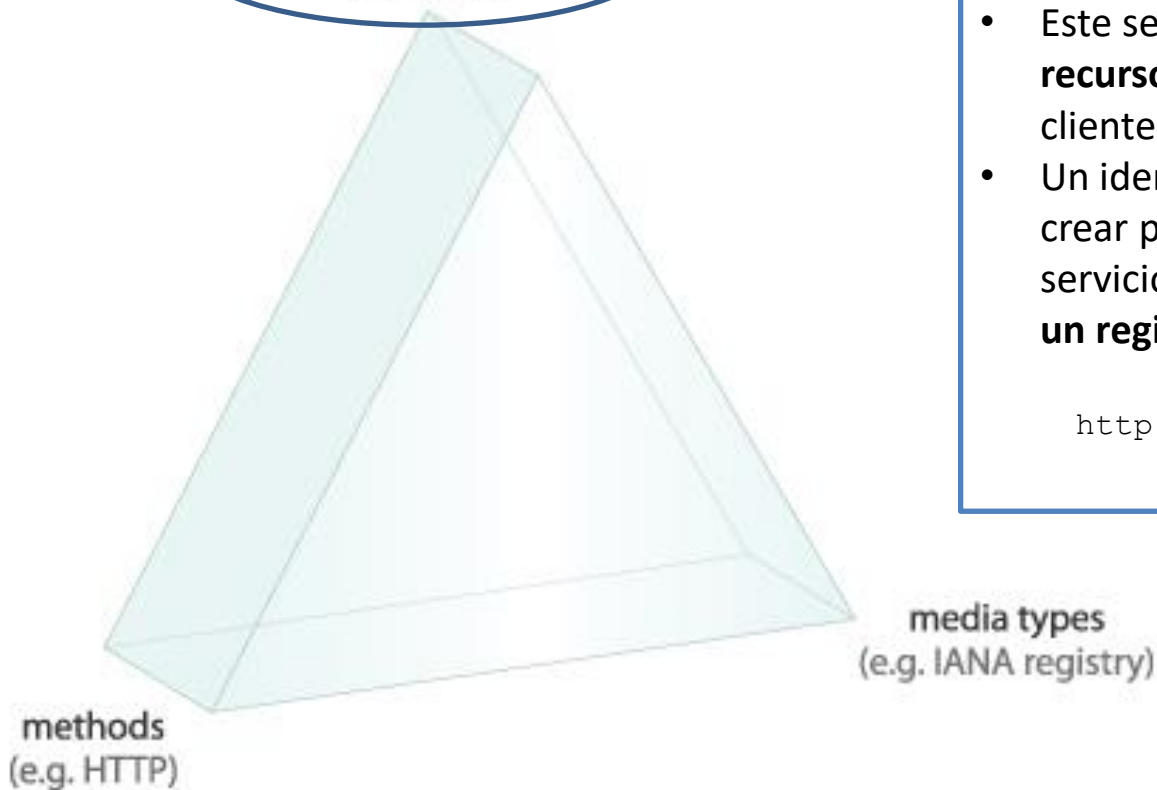
media types
(e.g. IANA registry)

Sintaxis de los identificadores de recursos

- Representan los recursos reales que un servicio expone.
- Un recurso puede ser :
 - De datos.
 - Procesamiento de lógica.
 - Archivos
 - O cualquier otra cosa que un servicio puede tener acceso.
- Un **identificador de recursos** es un identificador único asignado a uno o más recursos de los servicios.
- La **sintaxis** más común utilizada para expresar los identificadores de recursos uniformes es la sintaxis de la Web Resource Identifier (**URI**).

Metodología SOA aplicada REST

resource identifier syntax
(e.g. URL)

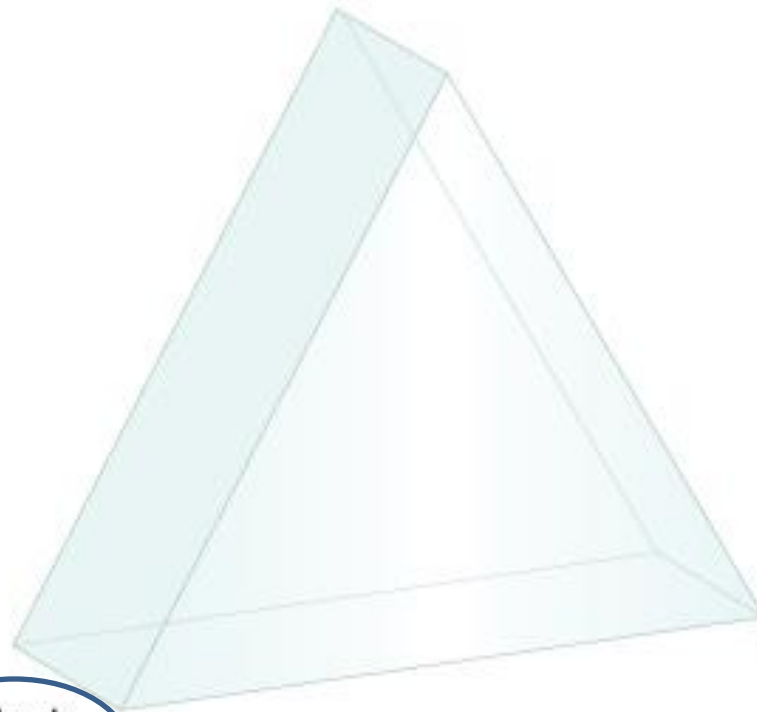


- Por ejemplo, podríamos tener un **servicio REST** entidad llamado **Cliente**.
- Este servicio se encargará de **encapsular recursos agnósticos**, relacionados con los clientes.
- Un identificador de recursos que podríamos crear para proporcionar la capacidad del servicio al cliente, para **recuperar datos de un registro específico del cliente**:

`http://cliente.mtis.org/cliente/C081`

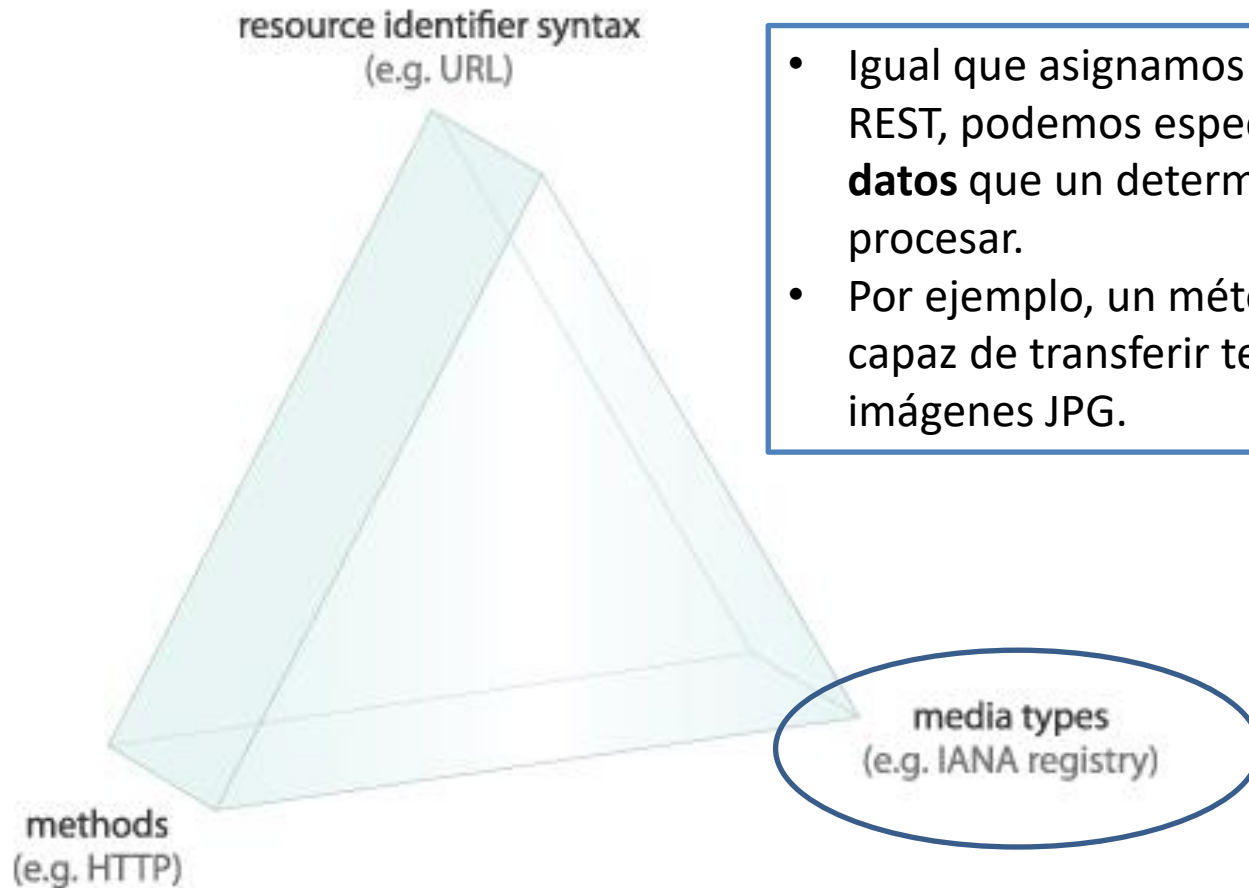
Metodología SOA aplicada REST

resource identifier syntax
(e.g. URL)

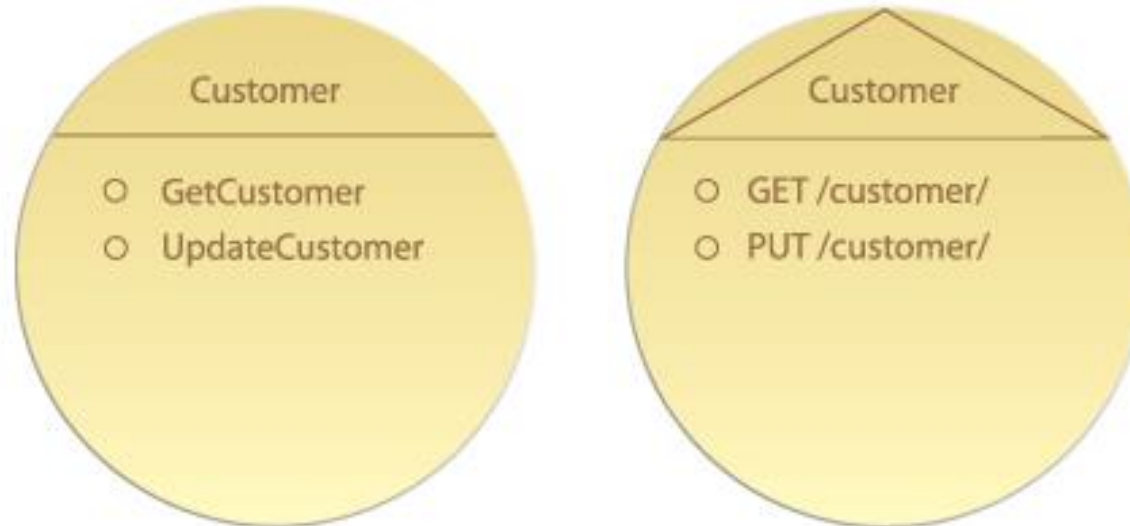


- Un **método** es un tipo de función proporcionada por un contrato uniforme para procesar identificadores y datos de recursos.
- Los métodos serán reutilizados a través de los servicios → tienden a ser muy **genéricos**.
- **HTTP** nos proporciona un conjunto de métodos genéricos, como **GET, PUT, POST, DELETE, HEAD** y **OPTIONS**, que se pre-definen en la especificación HTTP.
- Ejemplo:
`GET /cliente/C081 HTTP/1.1`

Metodología SOA aplicada REST

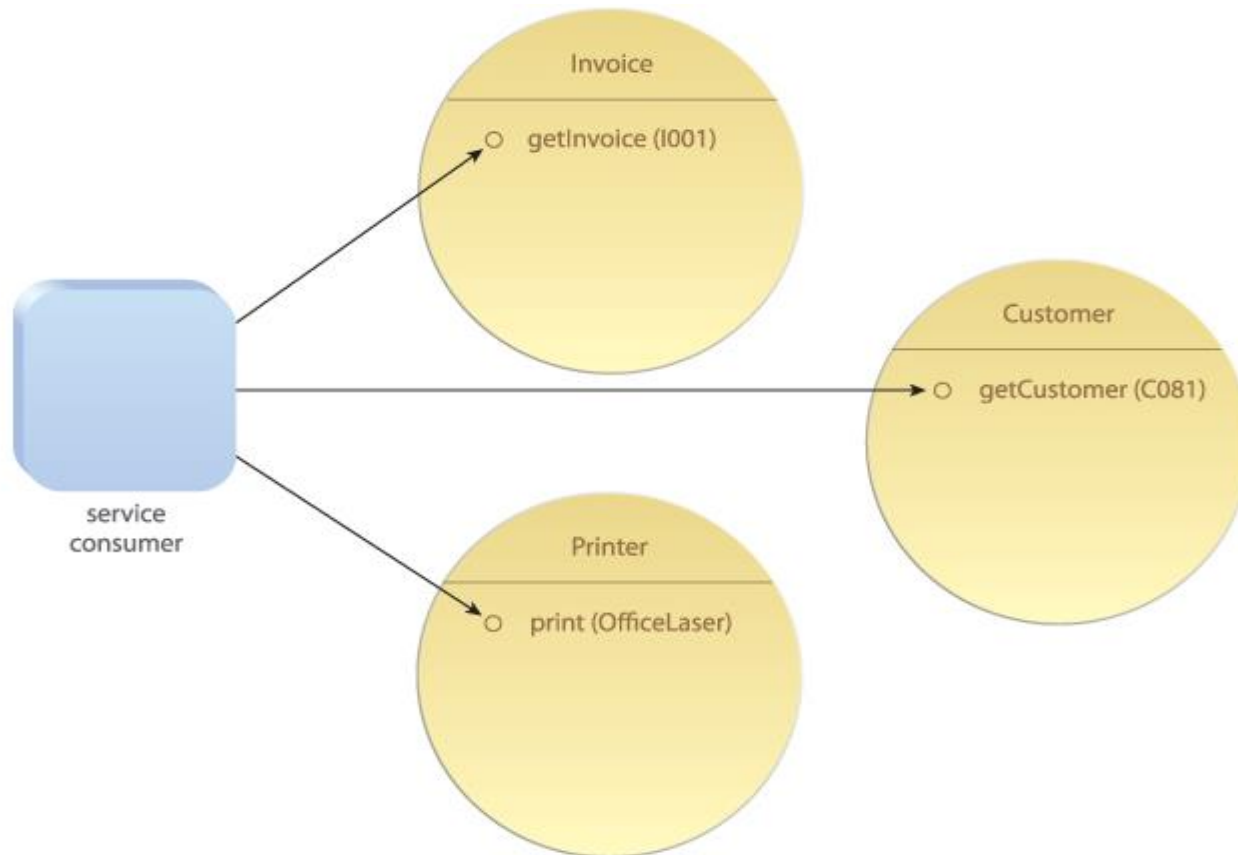


- Igual que asignamos métodos a los servicios REST, podemos especificar los **tipos de datos** que un determinado método puede procesar.
- Por ejemplo, un método GET puede ser capaz de transferir texto plano, XML, y las imágenes JPG.

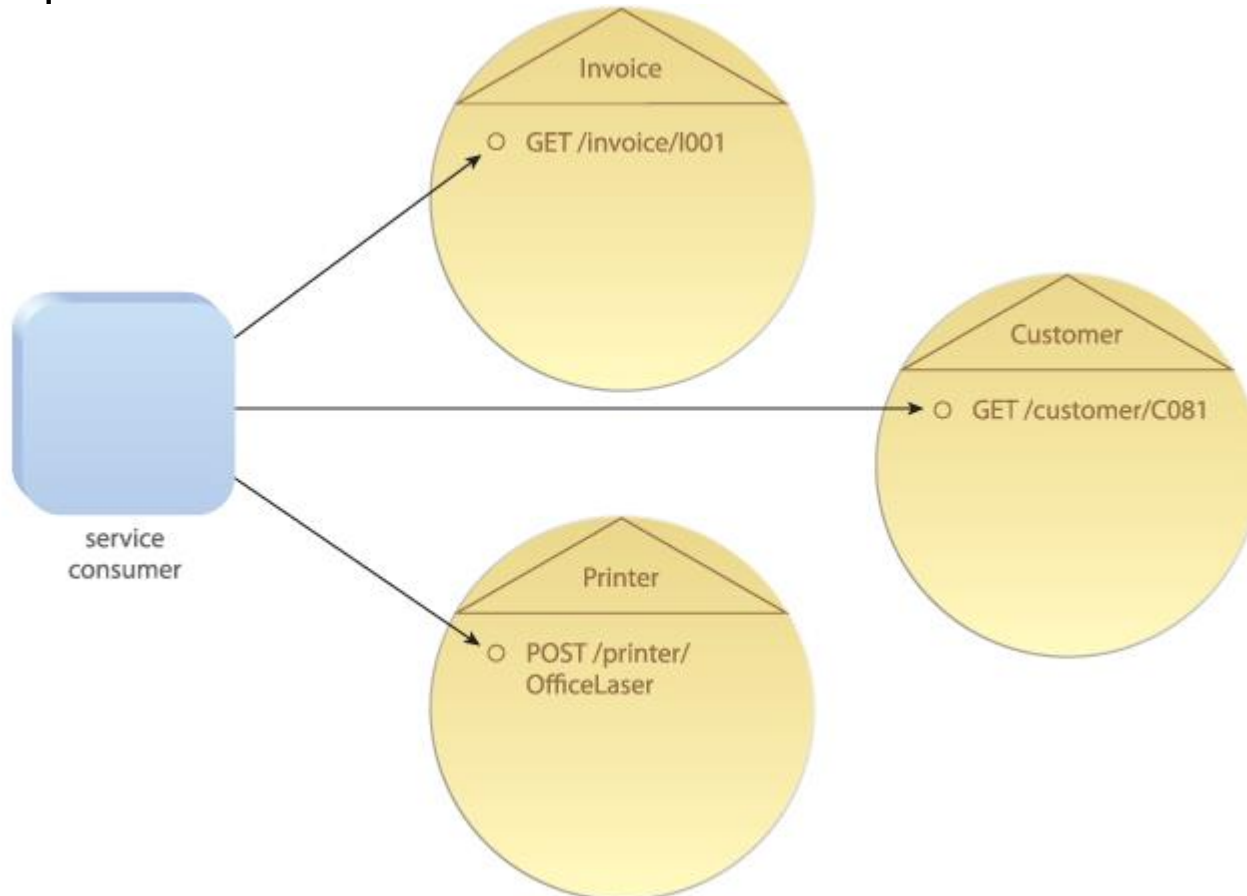


- Un contrato de servicio no **REST (izquierda)**, junto con un contrato de servicios **REST (derecha)**.
- Los contratos de servicios REST se denotan con una anotación de triángulo para indicar su dependencia en el contrato uniforme.

- Contratos Servicio REST vs Contratos de servicio no REST
 - Ejemplo Contrato Servicio **SOAP**



- Contratos Servicio REST vs Contratos de servicio no REST
 - Ejemplo Contrato Uniforme Servicio **REST**



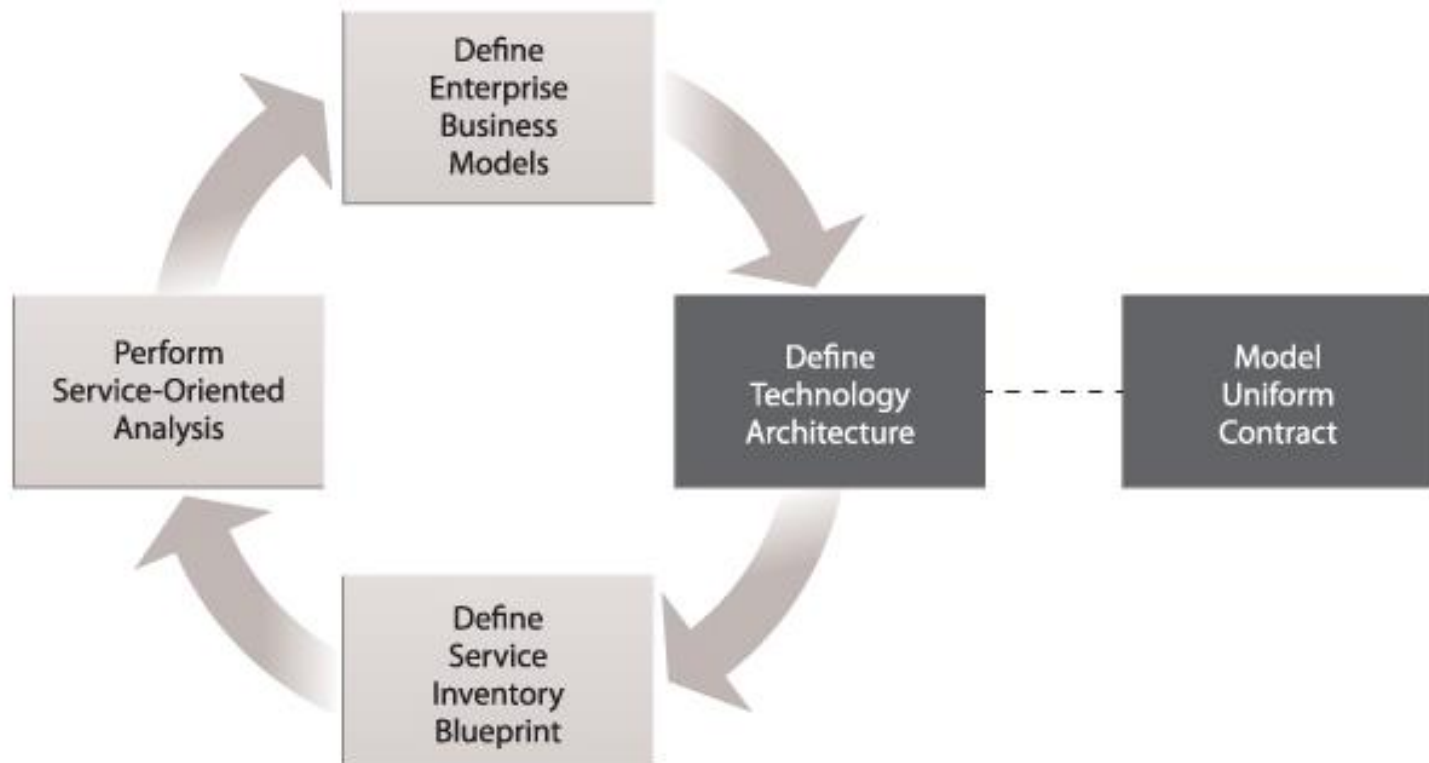
- **"SOA vs REST" o "SOA + REST"**
 - SOA y REST se describen comúnmente como **estilos arquitectónicos distintos**, cada uno con su propio enfoque de diseño que se lleva a cabo para alcanzar los objetivos específicos de diseño.
 - Puede parecer como si tengamos que elegir un estilo arquitectónico sobre el otro, en función de nuestras preferencias y objetivos individuales.
 - **SOA es el medio por el cual REST puede ser implementado.**

- **"SOA vs REST" o "SOA + REST"**
 - Comparaciones:
 - Las metas de SOA son estratégicas y centradas en el negocio.
 - Los **objetivos de REST** están **centrados** en la **tecnología** y pueden ayudar a alcanzar los objetivos de negocio estratégicos o tácticos.
 - La mayoría de los objetivos de diseño REST son directamente de apoyo a los objetivos de computación orientada a servicios.
 - **No hay conflictos** entre REST y metas computacionales orientadas a servicios.

- **“SOA vs REST” o “SOA + REST”**
 - REST tiene como objetivo establecer la arquitectura de aplicación que emula a la World Wide Web.
 - La **arquitectura REST** proporciona un **medio** por el cual **la arquitectura orientada a servicios** puede ser implementada.
 - La elección no es entre SOA y REST, sino más bien:
 - si REST es el medio de implementación correcto de una SOA, o ...
 - si la SOA es el modelo arquitectónico correcto por el cual una arquitectura REST debe formalizarse.

- **Análisis y modelado de servicios con REST**
 - El **modelado del contrato uniforme** está estrechamente **asociado** con la **etapa** de inventario de servicios de **análisis de proyectos**.
 - La definición inicial del **contrato uniforme** debe llevarse a cabo **antes** del **diseño de los contratos de servicios REST** porque los contratos de servicios REST, dependerán de las características de los contratos uniformes.
 - Un medio sencillo de incorporar la tarea de modelar el contrato uniforme, es con el ciclo de vida del servicio de análisis de inventario , combinándolo con *Definir la Arquitectura de la Tecnología* o establecerlo como un paso independiente.

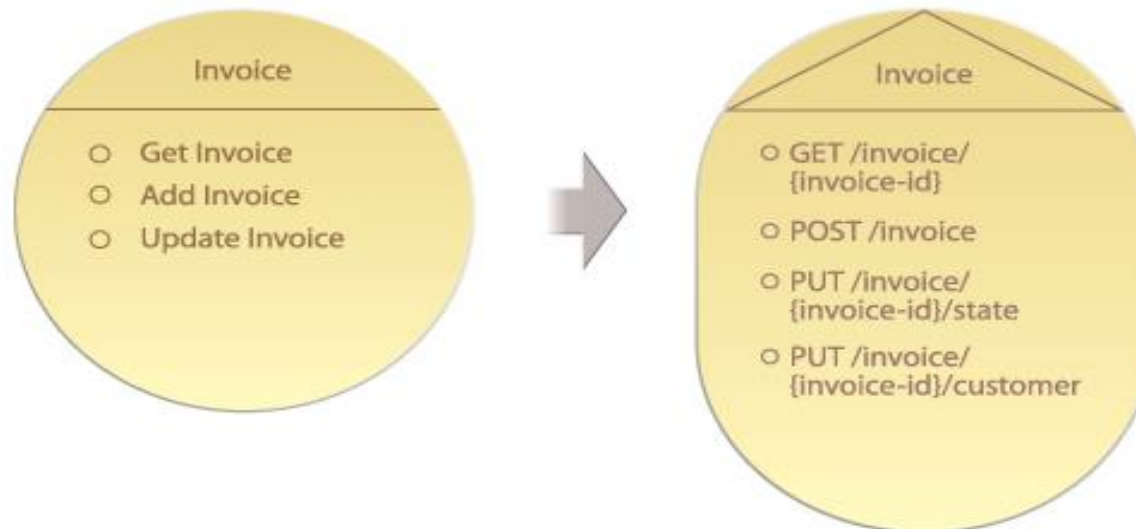
- **Análisis y modelado de servicios con REST**



- **Análisis y modelado de servicios con REST**
 - La necesidad de crear un servicio en particular puede provenir de **diversas fuentes y en diferentes circunstancias**.
 - Un servicio puede ser identificado:
 - Como necesario sobre la base de un inventario previo de servicios (**blueprint**), que asigna las funciones requeridas a los contextos funcionales que forman la base de los servicios existentes y futuros.
 - A través de solicitudes de cambio cuando se determina que es necesaria una **nueva funcionalidad**, y que esta nueva funcionalidad no tiene cabida en cualquier contexto de servicios funcionales existentes.
 - Mediante el análisis de los **sistemas heredados** aún no encapsulados dentro del alcance de un inventario de servicios.

• Análisis y modelado de servicios con REST

- Un candidato servicio REST puede ser modelado específicamente para incorporar características de los contratos uniformes → **Granularidad más fina.**



- El servicio candidato de actualización de la factura se divide en dos variantes,
 - una que actualiza el valor de estado de facturas .
 - Otro que actualiza el valor para el cliente de la factura.

• Análisis y modelado de servicios con REST

– Recursos frente a las Entidades:

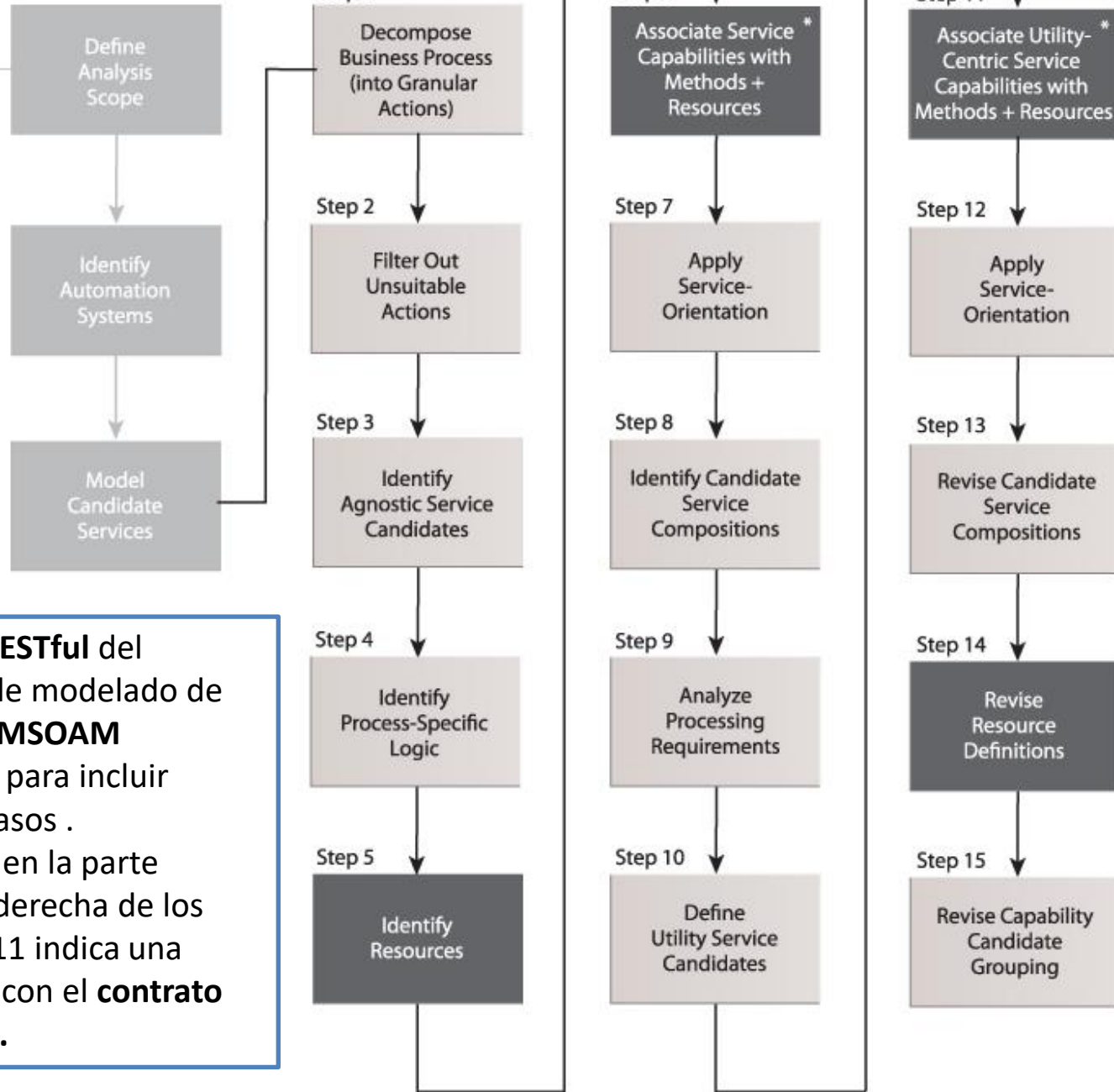
- Los Recursos/Entidades representan las "cosas" que necesitan ser accedidas y procesadas por los consumidores de servicios.
- ¿Cuál es entonces la diferencia entre un recurso y una entidad?
 - **Las entidades** están **centradas en el negocio** y se derivan de los modelos de negocio de la empresa, como los diagramas entidad relación, modelos de datos lógicos...
 - **Los recursos** pueden **estar centrados en el negocio o no** centrados en el negocio.
 - » Un recurso es cualquier "cosa" asociada con la lógica de automatización de negocios, habilitada por el inventario de servicios.
 - Las **entidades** se limitan normalmente a los artefactos y documentos de la empresa, tales como **facturas, reclamaciones, clientes, pedidos, etc...**

• Análisis y modelado de servicios con REST

– Recursos frente a las Entidades:

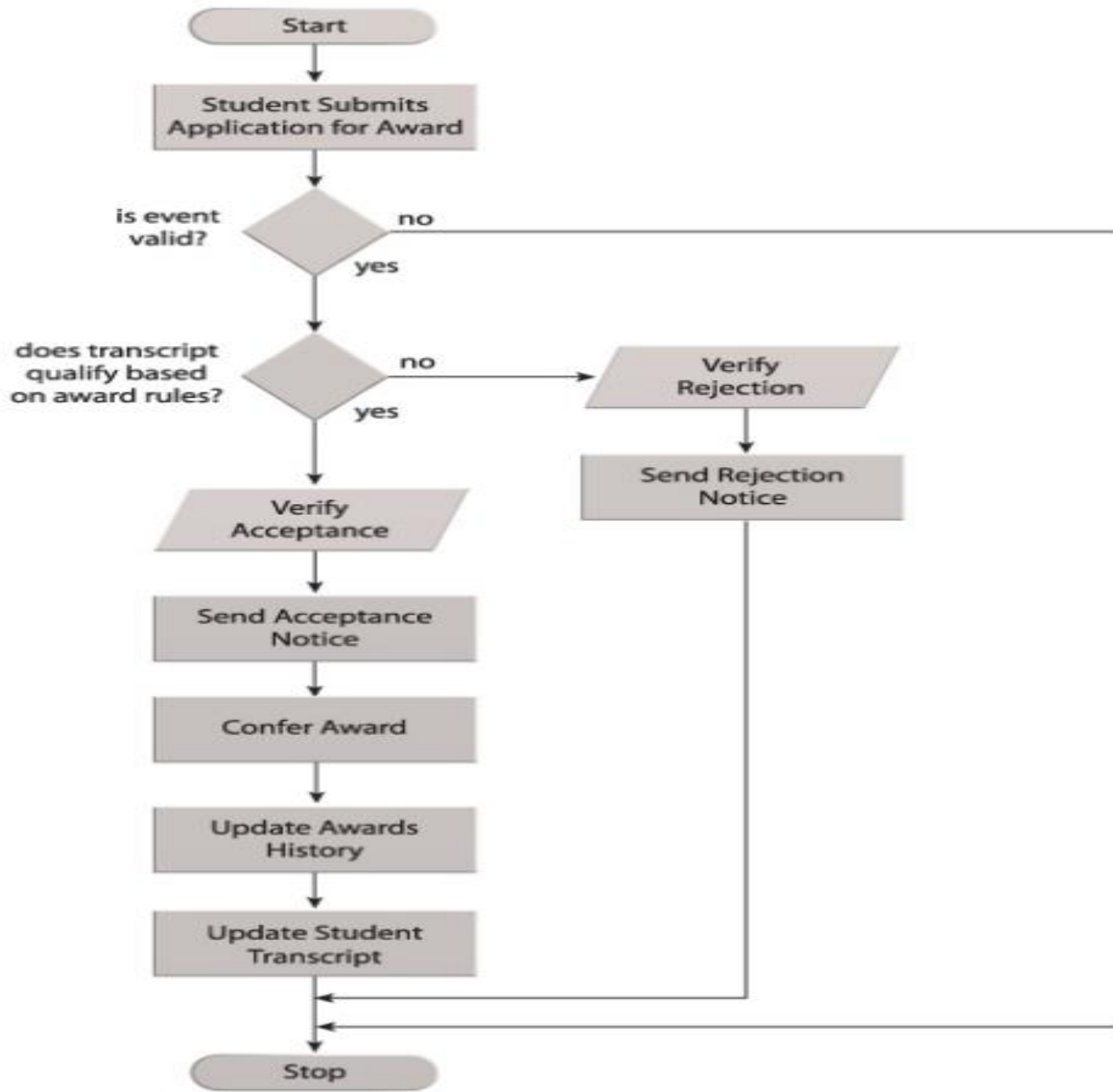
- Los Recursos/Entidades representan las "cosas" que necesitan ser accedidas y procesadas por los consumidores de servicios.
- ¿Cuál es entonces la diferencia entre un recurso y una entidad?
 - Algunas **entidades** son de **grano más grueso** que otras.
 - » Algunas entidades pueden encapsular otras.
 - » Ejemplo, una entidad de la factura puede encapsular un detalle entidad factura.
 - **Los recursos** también pueden variar en granularidad, pero a menudo **son de grano fino**.
 - Todas las entidades pueden relacionarse o estar basadas en los recursos.
 - » No todos los recursos se pueden asociar a las entidades, ya que algunos recursos son **non-business-centric**.

- **REST Modelado de Procesos**
 - El proceso proporcionando , es una variación de la corriente principal de modelado de servicios SOAP.
 - Contiene las modificaciones y medidas adicionales incorporados en apoyo del modelado de servicios REST.



- Versión **RESTful** del proceso de modelado de servicios **MSOAM** ampliado para incluir nuevos pasos .
- Asterisco en la parte superior derecha de los paso 6 y 11 indica una conexión con el **contrato uniforme**.

- **Ejemplo de Caso de Estudio**
 - MUA (Memorandum of Understanding and Agreement),: Reconocimiento de los logros académicos individuales.
 - MUA reúne un equipo de modelado de servicios compuesto por arquitectos SOA, SOA analistas y analistas de negocio.
 - El equipo comienza con un proceso de modelado de servicios REST para el proceso de negocio Atribución Premio al Logro Estudiantil.
 - Esta lógica de procesos de negocio representa los procedimientos seguidos para :
 - la evaluación,
 - la concesión,
 - y el rechazo de las solicitudes.



Metodología SOA aplicada REST

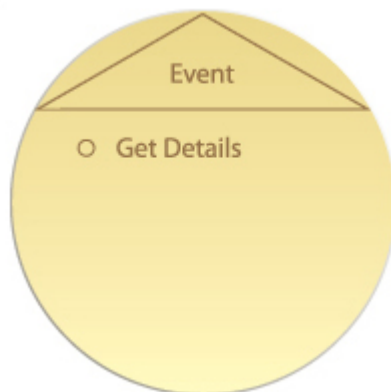
- **Paso 1: Descomposición Business Process (en acciones granulares)**
- El proceso original de negocios **La concesión de Premio Estudiantil** , se divide en las siguientes acciones granulares:
 - Iniciar Aplicación de Concesiones.
 - Obtener Detalles del evento.
 - Verificar Detalles del evento.
 - Si el evento es válido o no elegible para el premio, Cancelar Proceso.
 - Recibir Detalles de Premio.
 - Obtener expediente del estudiante.
 - Verificar que el expediente del estudiante cumple las bases y reglas de los premios.
 - Si el expediente del estudiante no cumple, Iniciar Rechazo.
 - Verificar manualmente Rechazo.
 - Imprimir Notificación Rechazo.
 - Enviar aviso de rechazo.
 - Verifique manualmente Aceptación.
 - Notificación de aceptación de impresión.
 - Enviar Notificación de Aceptación.
 - Conceder el Premio.
 - Guardar la concesión del premio en el expediente académico del estudiante.
 - Guardar la concesión del premio en la Base de datos de premios.
 - Imprimir una copia de atribución del premio.
 - Archivar una copia impresa de atribución del premio.

Metodología SOA aplicada REST

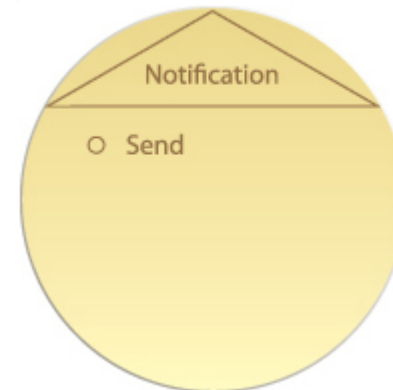
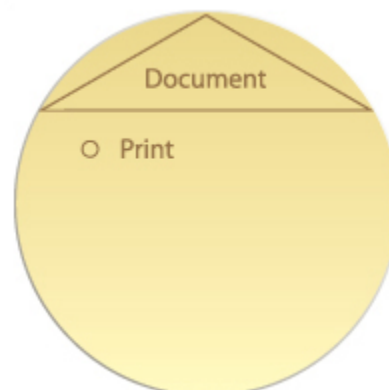
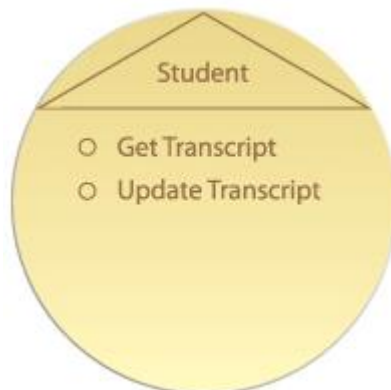
- **Paso 2: Filtro acciones inadecuadas**
- No toda la lógica de procesos de negocio es adecuada para la automatización y/o encapsulación para un servicio.
 - Iniciar Aplicación de Concesiones
 - Obtener Detalles del evento
 - Verificar Detalles del evento
 - Si el evento es válido o no elegible para el premio, Cancelar Proceso
 - Recibir Detalles de Premio
 - Obtener expediente del estudiante
 - Verificar que el expediente del estudiante cumple las bases y reglas de los premios
 - Si el expediente del estudiante no cumple, Iniciar Rechazo
 - **Verificar manualmente Rechazo**
 - Imprimir Notificación Rechazo
 - Enviar aviso de rechazo
 - **Verifique manualmente Aceptación**
 - Notificación de aceptación de impresión
 - **Enviar Notificación de Aceptación**
 - **Conceder el Premio**
 - Guardar la concesión del premio en el expediente académico del estudiante
 - Guardar la concesión del premio en la Base de datos de premios
 - Imprimir una copia de atribución del premio
 - **Archivar una copia impresa de atribución del premio**

- **Paso 3: Identificar Servicios Candidatos Agnósticos**
- Eliminamos del punto anterior los candidatos que son manuales y **destacamos** los que **no son agnósticos**:
 - Iniciar Aplicación de Concesiones.
 - Obtener Detalles del evento
 - Verificar Detalles del evento
 - Si el evento es válido o no elegible para el premio, Cancelar Proceso
 - Recibir Detalles de Premio
 - Obtener expediente del estudiante
 - Verificar que el expediente del estudiante cumple las bases y reglas de los premios.
 - Si el expediente del estudiante no cumple, Iniciar Rechazo.
 - Imprimir Notificación Rechazo
 - Enviar aviso de rechazo
 - Notificación de aceptación de impresión
 - Guardar la concesión del premio en el expediente académico del estudiante
 - Guardar la concesión del premio en la Base de datos de premios
 - Imprimir una copia de atribución del premio

- **Paso 3: Identificar candidatos Servicio Agnósticos**
 - Servicio Candidato Evento (Entidad)
 - Servicio Candidato Premios (Entidad)
 - Tres candidatos de capacidad de servicio, incluyendo dos que se basan en la misma acción.
 - Concesión
 - Actualización del Histórico



- **Paso 3: Identificar candidatos Servicio Agnósticos**
 - Servicio Candidato Estudiante (Entidad)
 - Obtener expediente del estudiante
 - Guardar la concesión del premio en el expediente académico del estudiante
 - Servicio Candidato Notificación (Utility)
 - Servicio Candidato Documento (Utility)



- **Paso 4: Identificar procesos específicos de la lógica**
 - Cualquier parte de la lógica de los procesos de negocio que queda después de completar el paso 3 será clasificada como **no-agnóstica** o **específicas para el proceso de negocio**.
 - Los tipos más comunes de las acciones que entran en esta categoría son:
 - Las reglas de negocio
 - La lógica condicional
 - La lógica excepción,
 - La lógica secuencia utilizada para ejecutar las acciones de procesos de negocio individuales.

- **Paso 4: Identificar procesos específicos de la lógica**
 - Si el expediente del estudiante no cumple, Iniciar Rechazo
 - Verificar Detalles del evento
 - Si el evento no es válido o no elegible para el premio, Cancelar Proceso
 - Verificar que el expediente del estudiante cumple las bases y reglas de los premios.
 - Si el expediente del estudiante no cumple, Iniciar Rechazo.
- Las acciones no se corresponden con los candidatos de capacidad de servicio.
 - Se identifican como la lógica que se produce internamente en el servicio de tareas **Concesión Premio Estudiante**.

- **Paso 4: Identificar procesos específicos de la lógica**
- Servicio Candidato Concesión Premio a Estudiantes (Tarea)
 - La acción Iniciar Aplicación de Concesiones, se traduce en un simple servicio de inicio de capacidad candidato.
 - Se espera que la capacidad de arranque será invocada por un programa de software por separado. *Iniciador de Composición*



• Paso 5: Identificar Recursos

- Identificamos recursos como:
 - Agnóstico (**multiusos**)
 - No agnóstico (**de uso individual**),
- Los **recursos agnósticos** se pueden incorporar en servicios agnósticos y candidatos de capacidad sin limitación.
- El **beneficio** para la identificación de recursos agnóstico es destinar como partes de la empresa que son **susceptibles de ser compartidos y reutilizados** con más frecuencia que los recursos no agnósticos.

- **Paso 5: Identificar Recursos**
- Se identifican los siguientes recursos potenciales:
 - / Proceso / → / Proceso Otorgamiento Premio /
 - / Aplicación / → /Aplicación de Concesión/
 - / Evento /
 - / Premio /
 - / Expediente Académico Estudiante/
 - / Aviso remitente /
 - / Impresora /

• Paso 5: Identificar Recursos

- Mapear las entidades empresariales a los recursos.

Entidad	Recurso
Evento	/Event/
Premio	/Premio/
Estudiante	/Expediente Académico Estudiante/

- Los recursos adicionales no se asignan, ya que no se refieren actualmente a las entidades de negocios conocidas.
 - Pueden llegar a ser asignados en futuras iteraciones del proceso de modelado de servicios.

- **Paso 6: Asociar Capacidades de servicio con Recursos y Métodos**
 - Asociamos a los candidatos de capacidad de servicio definidos en los pasos 3 y 4 con los recursos definidos en el Paso 5.
 - Así como con los métodos de contratos uniformes disponibles que pueden haber sido establecidos.
 - En esta etapa, es común **asociar acciones** con **métodos HTTP regulares**, tal como se define a través del modelado del contrato uniforme.

- **Paso 6: Asociar Capacidades de servicio con Recursos y Métodos**

- Servicio Candidato Concesión de Premios a Estudiantes (Tarea)
 - El documento de negocios requiere que la entrada principal para dar inicio al proceso sea la solicitud presentada por el estudiante.



- Inicialmente se asumió que el recurso [/Aplicación de Concesión/](#) se necesitaría para representar este documento.
 - Sin embargo, tras un análisis más detallado, resulta que con un método POST para enviar el documento de solicitud del recurso , es suficiente.

- **Paso 6: Asociar Capacidades de servicio con Recursos y Métodos**
 - Servicio Candidato Evento (Entidad)
 - El servicio candidato de capacidad **Obtiene Detalles** se agrega con el método GET y el recurso / **Evento** /



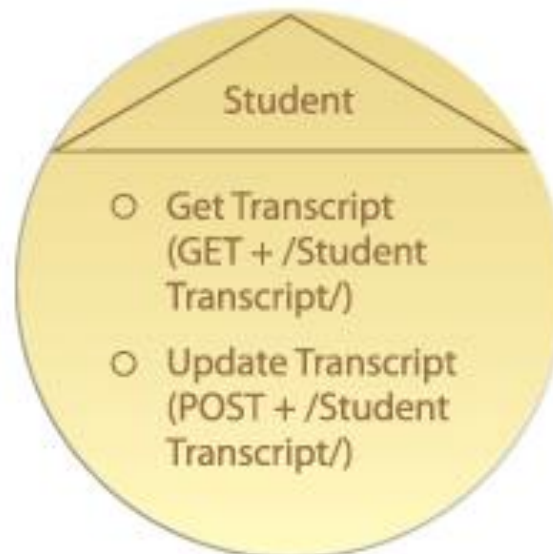
- **Paso 6: Asociar Capacidades de servicio con Recursos y Métodos**
 - Servicio Candidato Premio (Entidad)
 - El Servicio Candidato de Capacidad, **Obtener Detalles** se asocia con el método GET del recurso [/Premio/](#)
 - Los servicios candidatos de capacidad **Concesión** y **Actualizar Historico** requieren datos de entrada para actualizar los recursos , con lo que se amplia con un método POST el recurso [/Premio/](#)



• Paso 6: Asociar Capacidades de servicio con Recursos y Métodos

– Candidato de Servicio Estudiante (Entidad)

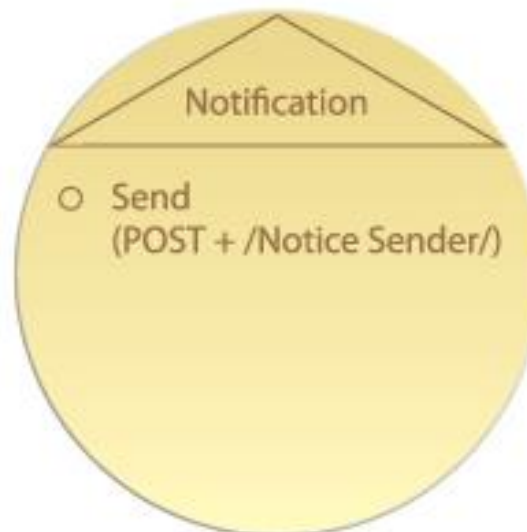
- El servicio candidato de capacidad **Obtener expediente alumno** se asocia con el método GET y el recurso [/Expediente estudiante/](#).
- El servicio candidato de capacidad **actualización expediente estudiante** se adjunta con el método POST al recurso [/Expediente estudiante/](#).



- **Paso 6: Asociar Capacidades de servicio con Recursos y Métodos**

- Candidato Servicio Notificación (Utility):

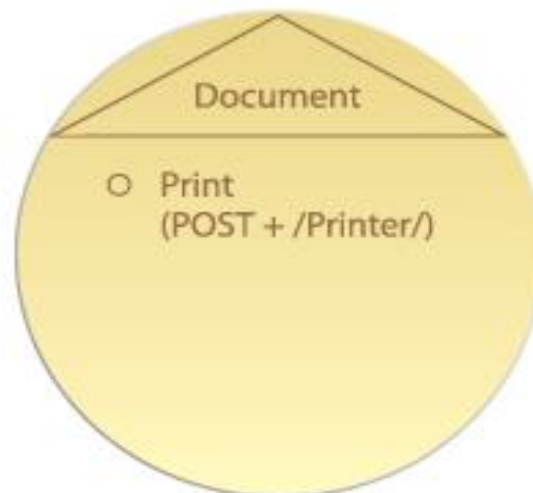
- El servicio candidato de capacidad Enviar se amplía con el método POST y el recurso [/ Aviso remitente /](#)



- **Paso 6: Asociar Capacidades de servicio con Recursos y Métodos**

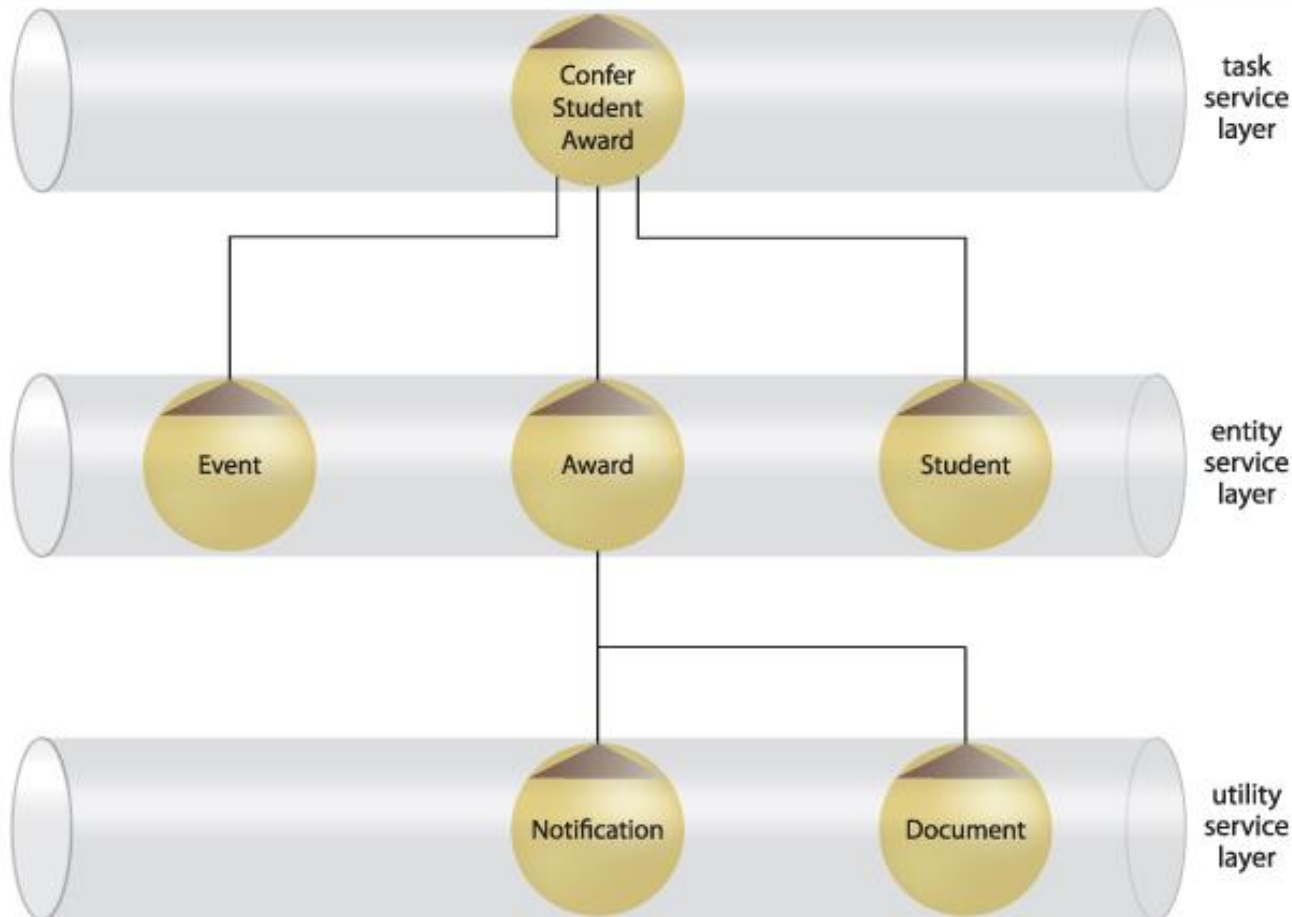
- Servicio Candidato Documento (Utility)

- El servicio candidato de capacidad de **impresión**, se amplía con un método POST y el recurso [/ Impresora /](#).
 - Cualquier documento enviado a la capacidad de impresión será publicado en el recurso [/ Impresora /](#) e impreso.



- **Paso 7: Aplicar la Orientación a Servicios**
- **Paso 8: Identificar Composiciones Servicio candidatos**
 - Documentamos las interacciones de capacidad de servicio más comunes que pueden tener lugar durante la ejecución de la lógica de los procesos de negocio.
 - En esta etapa comenzamos a tomar una mirada más cercana a las **necesidades de intercambio de datos**
 - Los **servicios** que componen entre sí, tienen que **intercambiar datos**.

Metodología SOA aplicada REST

Jerarquía de composición

- En cada caso, la tarea de **Concesión Premio Estudiante** invoca el evento, Premio, y estudiante.
- El servicio de entidad **Premio** se compone de el servicio público de notificación para emitir la aceptación o rechazo de las notificaciones y, si el premio es conferido, el servicio de la utilidad del documento para imprimir el registro de adjudicación.

- La siguiente serie de pasos es opcional y más adecuado para los procesos de negocio complejos y grandes arquitecturas de inventario de servicios:
 - Paso 9: Analizar Requisitos de procesamiento
 - Paso 10: Definir Candidatos Servicio Utility
 - Paso 11: Asociados Capacidades Utility-Centric servicio con recursos y métodos
 - Paso 12: Aplique la Orientación a Servicios
 - Paso 13: Revisar candidatos Composiciones Servicio
 - Paso 14: Definiciones de Recursos Revisar
 - Paso 15: Agrupación Candidato Capacidad Revisar

- **Diseño Orientado a Servicios.**
 - Por último nos quedaría el diseño orientado a servicios
 - Tendríamos dos tareas en particular:
 - El **diseño** de un **contrato uniforme** para un inventario de servicios.
 - El **diseño** de los **contratos de servicios individuales** dentro del inventario de servicios y en el cumplimiento del contrato uniforme.

Integración de Rest en BPEL

Integración de Rest en BPEL

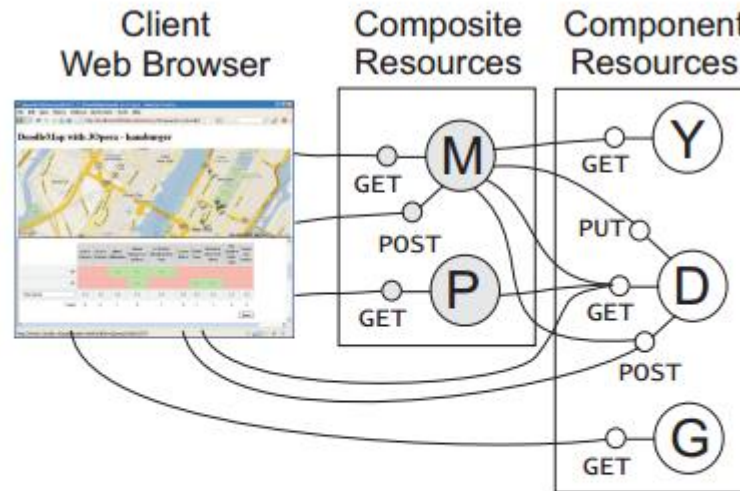
- La integración de Rest en BPEL, es posible mediante los siguientes métodos:
 - BPEL 2.0
 - WSDL 1.1 Extensions for REST
- Existe una tercera alternativa para la composición de servicios Rest:
 - JOpera for Eclipse

- WSDL 1.1 Extensions for REST
 - El tradicional HTTP binding en WSDL solo soportaba las operaciones GET y POST.
 - Rest , necesita las operaciones adicionales PUT y DELETE.
 - Esto lo logramos con: **WSDL 1.1 Extensions para REST**
 - <http://ode.apache.org/extensions/wSDL-1.1-extensions-for-rest.html>
 - Ejemplo paso a paso:
 - <http://waruapz.blogspot.com.es/2014/02/rest-service-with-ode-process.html>

- JOpera for Eclipse

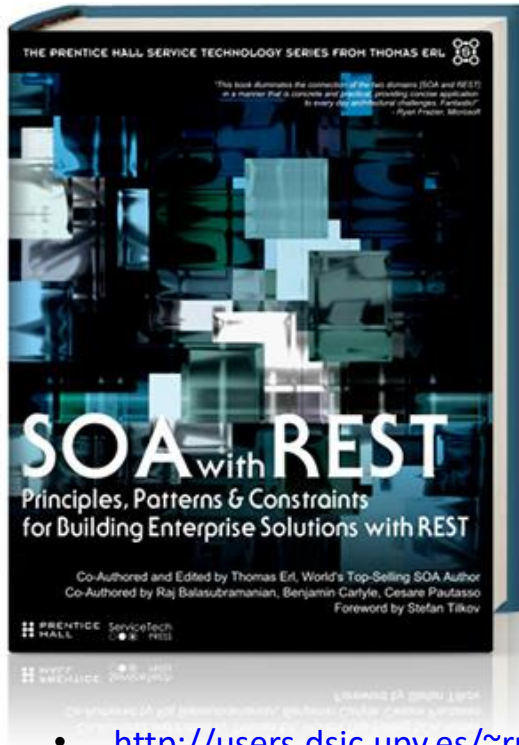
- Es una herramienta rápida de composición, visual y construcción de servicios.

- <http://www.ibm.com/developerworks/java/j2ee/102303soa.html>



Legend:

Resource	RESTful Service
Y	Yahoo! Local Search
G	Google Maps API
D	Doodle API
M	DoodleMap Mashup
P	DoodleMap Poll State Proxy



SOA with REST

Principles, Patterns & Constraints for Building Enterprise Solutions with REST

Thomas Erl

ISBN 0137012519

2012, Prentice Hall

- <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>
- <http://pensandoensoa.com/2011/03/19/869>
- <http://www.w3.org/Submission/wadl/#x3-40001.3>
- <http://www.genbetadev.com/programacion-en-la-nube/define-el-diseno-de-tus-apis-con-blueprint-raml-o-swagger>
- <http://blog.gfi.es/swagger-documentando-servicios/>