MTIS

SOA
Guía Buenas Prácticas

Profesor: Alejandro Sirvent Llamas

- La **incorporación** de los **principios** de **diseño** orientados a servicios en normas formales, es **fundamental** para el **éxito** de SOA en una organización.
- En esta guía se proporcionan un **conjunto de directrices**, que pueden ser utilizadas como un punto de partida desde el que se pueden derivar nuestras propias normas.

- Aplicar estándares de nomenclatura
  - El Etiquetado de los servicios es el equivalente al etiquetado de la infraestructura de TI.
    - Por tanto, es esencial que las **interfaces de servicios** sean **auto-descriptivos**.
  - Por lo tanto, las normas de nomenclatura deben ser definidos y aplicados a:
    - service endpoint names
    - service operation names
    - message values

- Aplicar estándares de nomenclatura
  - Convenciones de nomenclatura existentes varían según la organización.
    - Unos emplean OO donde los objetos se asignan con nombres y los métodos se etiquetan con verbos.
    - Otros simplemente aplican verbos para ambos componentes y sus métodos.
  - No existe una norma de denominación perfecta para todas las organizaciones.
  - **La clave es que lo que se decida aplicar, debe aplicarse consistentemente a toda la solución orientada a servicios.**

- Aplicar estándares de nomenclatura
  - Sugerencias:
    - Los servicios candidatos con **alto potencial de reutilización** entre aplicaciones, no deben de tener características de nombres que hagan alusión a los procesos de negocio, para el que fueron construidos.
      - GetTimesheetSubmissionID, reducirlo a GetTimesheetID o simplemente getId.
    - Los **servicios de aplicaciones** tienen que ser nombrados de acuerdo con el contexto de procesamiento en virtud del cual sus operaciones se agrupan.
      - Ejemplos simplificados de nombres de servicios de aplicación adecuados son:
        - » CustomerDataAccess, SalesReporting, y GetStatistics.

- Aplicar estándares de nomenclatura
  - Sugerencias:
    - Las operaciones de **servicios de aplicaciones** necesitan comunicar con claridad la naturaleza de su funcionalidad.
      - Ejemplos de nombres adecuados de operación de servicios de aplicación son:
        - » GetReport, ConvertCurrency y verifydata.
    - Servicios **Entity-centric** , necesitan mantener la representación de los modelos de entidad , que provienen de los candidatos de servicios correspondientes.
      - Las denominaciones usadas deben reflejar las establecidas en los modelos de entidades originales de la organización.
      - Normalmente, este tipo de servicio utiliza el sustantivo , nombrando sólo la estructura.
      - Ejemplos:
        - » Factura, Cliente y Empleado.

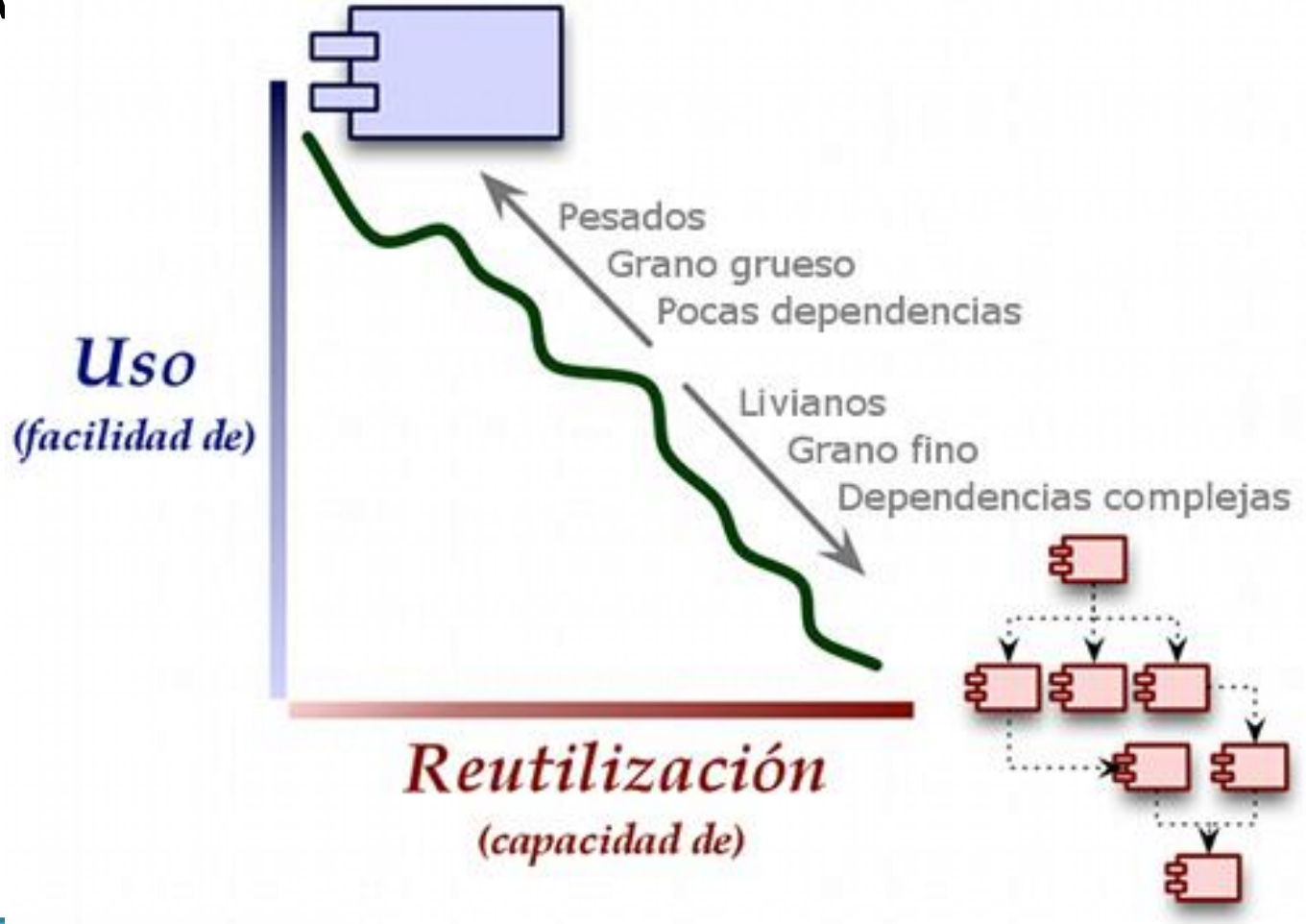
- Aplicar estándares de nomenclatura
  - Sugerencias:
    - Las **operaciones** de servicio para los servicios **entity-centric**, deben basarse en verbos y no deben repetir el nombre de la entidad.
      - Por ejemplo, un servicio de entity-centric, llamado factura **no debería tener una operación denominada AddFactura**.

- Aplicar un adecuado nivel de granularidad
  - La granularidad en la que los servicios pueden ser diseñados puede variar.
  - ***El tener múltiples funciones agrupadas en una sola operación , puede ser indeseable para los solicitantes que sólo requieren el uso de una de esas funciones .***
  - La granularidad de la interfaz de servicio es un punto de decisión estratégica clave, que merece una gran atención durante la fase de diseño orientado a servicios .

- Aplicar un adecuado nivel de granularidad
  - Pautas para hacer frente a este problema:
    - **Comprender** totalmente las **limitaciones** de **rendimiento** del entorno de implementación de destino y explorar tecnologías alternativas de apoyo.
    - Explorar la posibilidad de **proporcionar** WSDL (**grano grueso y menos grueso**) , alternando definiciones de los mismos servicios Web.
      - O explorar la opción de suministrar operaciones redundantes de **varias granularidades** en la misma definición WSDL.
      - Estos enfoques **desnormalizan** los contratos de servicio, pero **abordan** los **problemas** de **rendimiento** y dan cabida a una amplia gama de peticiones.

- Aplicar un adecuado nivel de granularidad
  - Pautas para hacer frente a este problema:
    - Asignar las interfaces de **grano grueso** a los servicios designados como **puntos finales de la solución.**
    - Permitir las interfaces de **grano más finos** para los servicios confinados a los **límites predefinidos**.
    - Maximizar la reutilización complica la utilización.
      - la **reutilización** se favorece en los servicios de **grano finos**.
      - La **interoperabilidad** se promueve en servicios de **grano grueso**.

- Aplicar un adecuado nivel de granularidad
  -



icios

os

.

inos.

no

- Las operaciones de diseño de servicios deben ser intrínsecamente extensibles
  - Independiente de lo bien diseñados que estén los servicios en el primer despliegue → nunca estarán totalmente preparados para el futuro.
  - Se producirán cambios en los procesos de negocio → **servicios serán extendidos**.
  - Dependiendo de la naturaleza del cambio , a veces se podrá lograr la extensibilidad sin romper la interfaz de servicio existente.
  - Es importante **diseñar** las operaciones y los mensajes en la medida de lo posible, **como actividades agnósticas**.

- Identificar los solicitantes del servicio conocidos y potenciales.
  - Los **servicios** son casi siempre construidos como **parte** de la entrega de una **solución de automatización específica**.
    - Por lo tanto , están diseñados para hacer frente a los requerimientos del negocio , ya que pertenecen a la aplicación .
  - **Limitar** el diseño de servicios para satisfacer las necesidades inmediatas **puede inhibir su potencial como reutilizable, adaptable , e interoperable**.

- Identificar los solicitantes del servicio conocidos y potenciales.
  - Es **aconsejable** que cualquier proceso de diseño de los servicios existentes, **incorpore un análisis especulativo de lo que el servicio pueda ser utilizado fuera** de sus **límites** iniciales de aplicación.
  - Puede ser **útil determinar** los **posibles futuros solicitantes del servicio**, para posteriormente incorporar sus necesidades previstas en el diseño de servicio actual .

- Considerar el uso de documentos WSDL-OpenApi modulares.
  - Los **WSDL**, se pueden **modularizar**, separando en varios ficheros (mediante *import*), los diferentes elementos de una definición de servicio.
    - Se pueden **ensamblar** de forma dinámica **en tiempo de ejecución** a través de la utilización de las declaraciones de importación que enlazan con archivos separados que contienen partes de la definición del servicio.
  - **OpenApi**, permite mediante la directiva *component*, modularizar y reutilizar de igual manera a WSDL.

- Considerar el uso de documentos WSDL-OpenApi modulares.
  - WSDL: Esto **permite definir** los **módulos** para los **tipos**, **operaciones** y **enlaces** que pueden ser compartidos a través de documentos WSDL.
    - También nos **permite aprovechar** los módulos de esquema **XSD existentes**, que ya tengamos diseñados.
  - OpenAPI: Permite reutilizar parámetros, respuestas, ejemplos, esquemas de seguridad, cabeceras, etc...

- Considerar el uso de documentos WSDL modulares.(Ejemplo)

```
<definition ...>
<schema>
<orders>
...
</orders>

<invoice>
...
</invoice>
<shipments>
...
</shipments>
</schema>
...
</definition>
```

```
<definition ...>
<schema>
<import namespace="..." schemalocation="orders.xsd">
<import namespace="..." schemalocation="invoice.xsd">
<import namespace="..." schemalocation="shipment.xsd">
...
</schema>
...
</definition>
```

Ventajas:

- Es fácil de gestionar, cuando aumenta la complejidad.
- No hay conflictos de nombres, ya que cada xsd importado tiene su propio espacio de nombres.
- Diseño claro y limpio

Inconvenientes:

- Leer elementos con múltiples namespace.

```
<import namespace="http://.../common/wsdl/"
   location="http://.../common/wsdl/bindings.wsdl"/>
```

- Considerar el uso de documentos OpenApi modulares.(Ejemplo)

```yaml
components:
  #------------------------------
  # Reusable schemas (data models)
  #------------------------------
  schemas:
    User:           # Can be referenced as '#/components/schemas/User'
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string

    Error:          # Can be referenced as '#/components/schemas/Error'
      type: object
      properties:
        code:
          type: integer
        message:
          type: string
```

```yaml
components:
  # Reusable schemas (data models)
  schemas:
    ...

  # Reusable path, query, header and cookie parameters
  parameters:
    ...

  # Security scheme definitions (see Authentication)
  securitySchemes:
    ...

  # Reusable request bodies
  requestBodies:
    ...

    ...

  # Reusable links
  links:
    ...

  # Reusable callbacks
  callbacks:
    ...
```

Ventajas:

- Es fácil de gestionar, cuando aumenta la complejidad.
- No hay conflictos de nombres, ya que cada componente tiene su propio espacio de nombres.
- Diseño claro y limpio.

- Usar los **namespace** cuidadosamente
  - Una definición WSDL se compone de una colección de elementos con diferentes orígenes.
  - Cada definición implicará una serie de espacios de nombres diferentes (namespace).

- Usar los **namespace** cuidadosamente
  - En el montaje de un WSDL modular, los **namespace** adicionales entran en juego, sobre todo cuando se importan las definiciones de esquema XSD.
  - Además, al definir sus propios elementos, se pueden establecer más **namespace** para representar partes específicas de los documentos WSDL.

- Usar los **namespace** cuidadosamente
  - Es muy recomendable organizar el uso de **namespace** en las WSDL.
  - El WS-I requiere el uso del atributo **targetNamespace** para asignar un espacio de nombres para el WSDL en su conjunto.
    - Si el esquema XSD se encuentra embedido dentro de la definición WSDL → el WS-I exige que también se le puede asignar un valor **targetNamespace** (que puede ser el mismo valor utilizado por el WSDL).

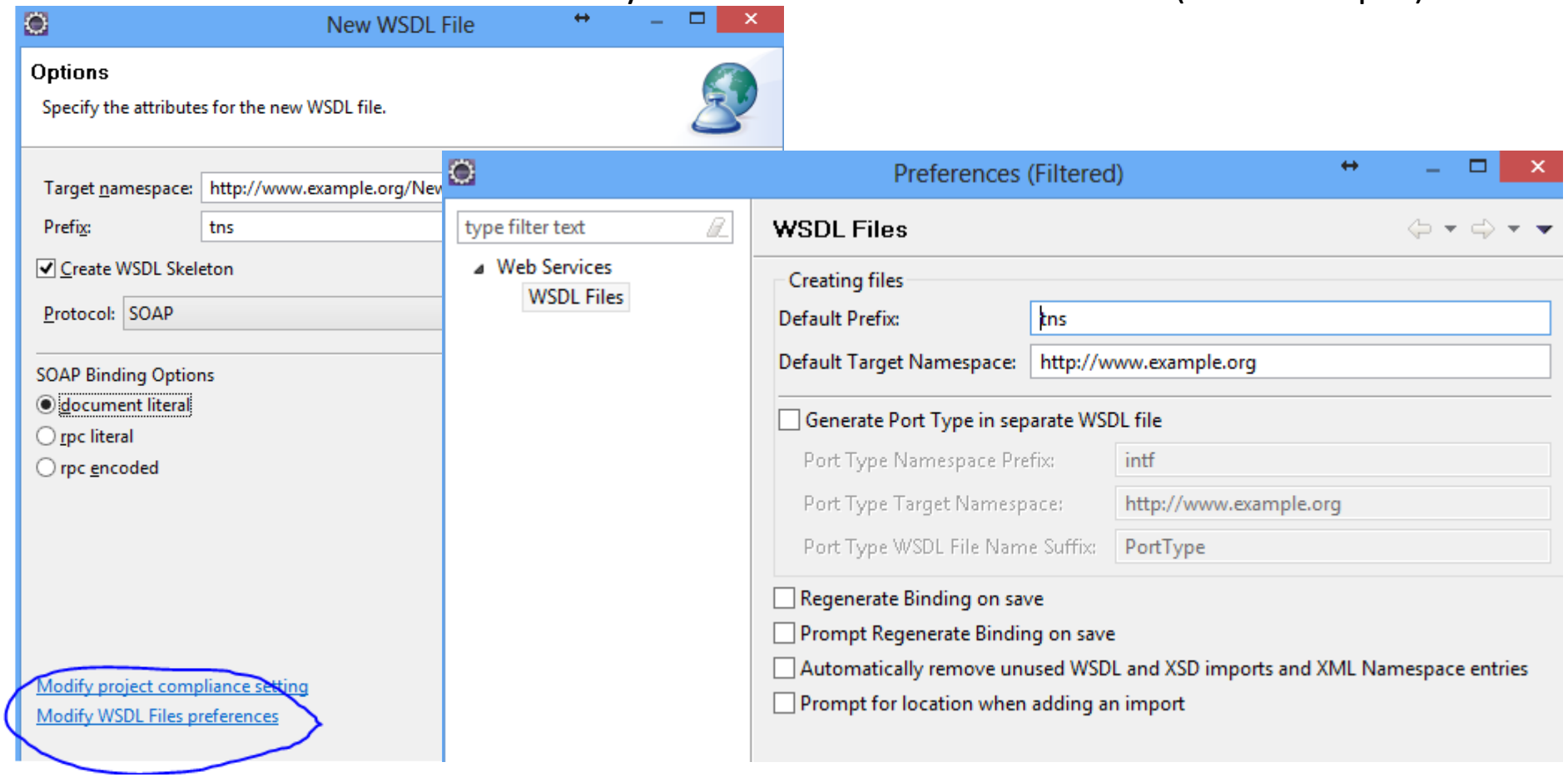
- Utilice el documento SOAP y los valores de atributos literales
  - Hay dos atributos específicos que establecen el formato de carga útil del mensaje SOAP y el sistema de tipo de datos utilizado para representar los datos de carga útil.
    - Estos son los atributos de estilo (**style**) usado por **soap:binding** y el atributo de uso (**use**) asignado al elemento **soap:body**. Estos dos elementos se encuentran dentro de la construcción de WSDL **binding**.

- Utilice el documento SOAP y los valores de atributos literales
  - El Cómo se establecen estos atributos es significativo, ya que se relaciona con la manera en la que el contenido del mensaje SOAP, está estructurado y representado:
    - El atributo de estilo se le puede asignar un valor de "**document**" o " **rpc** ".
      - El primero apoya la incorporación de documentos XML completos en el SOAP body.
      - El segundo está diseñado más para reflejar la comunicación RPC tradicional.

- Utilice el documento SOAP y los valores de atributos literales
  - El Cómo se establecen estos atributos es significativo, ya que se relaciona con la manera en la que el contenido del mensaje SOAP, está estructurado y representado:
    - El atributo de uso se puede ajustar a un valor de “**literal**" o "**codificado**".
      - SOAP originalmente proporcionó su propio sistema de tipo, para representar el contenido del cuerpo .
      - Más tarde, se incorporó soporte para tipos de datos XSD . **El valor de este atributo indica que tipo de sistema queremos utilizar para los mensajes**.
        - » Los " **literales**" de ajuste establecen que se aplicarán los tipos de datos XSD .

- Utilice el documento SOAP y los valores de atributos literales
  - Considerando los dos atributos anteriores y sus valores obtenemos las siguientes combinaciones soportados por SOAP:
    - style:RPC + use:encoded
    - style:RPC + use:literal
    - style:document + use:encoded
    - style:document + use:literal
  - El estilo **style:document + use:literal** : es la combinación preferida por SOA → apoya la noción de modelo de mensajería al estilo de documento, clave para la realización de las características de las especificaciones fundamentales de WS -*.
  - El **WS- I Basic** requiere que el atributo "**use**" siempre sea "**literal**".

- Utilice el documento SOAP y los valores de atributos literales (uso en eclipse)



- Emplear WS-I Profiles, aúnque no se requiera el cumplimiento de WS-I.
  - Se recomienda que se considere el uso de los estándares y buenas prácticas proporcionadas por WS-I Profiles.
    - Son sólidos, están bien investigados y probados y se puede ahorrar una gran cantidad de tiempo y esfuerzo en el desarrollo de estándares propios de diseño.
    - **WS-Security** (**Seguridad en Servicios Web**), no visto en SOA, pero importante.

- Document Service con metadatos
  - WS-Policy (W3C) y WS-MetadataExchange (BEA Systems, IBM, Microsoft, SAP) presentan un papel importante en la calidad y profundidad de las descripciones de servicio.
  - Las políticas en particular, representan un complemento importante de metadatos en las definiciones WSDL.
    - Ejemplo:
      - Una política puede expresar ciertos requisitos de seguridad, las preferencias de procesamiento y características de comportamiento de un proveedor de servicios.
      - Esto permite a los solicitantes del servicio, evaluar mejor a un proveedor de servicios y les ofrece la oportunidad de estar plenamente preparados para la interacción.

- Document Service con metadatos
  - **Un servicio debe ser documentado** para comunicar fácilmente un requisito de servicio, características o restricciones a otros que quieran utilizarlo.
  - La información se puede añadir a una definición de WSDL mediante el uso del elemento "**documentation"** , e incluso podría estar contenido dentro de un documento de metadatos que se publica por separado.
  - **Esto promueve el descubrimiento y la reutilización de los servicios**.

Service-Oriented Architecture: Concepts, Technology, and Design

By Thomas Erl

.....

Publisher: **Prentice Hall PTR**
Pub Date: **August 04, 2005**
ISBN: **0-13-185858-0**
Pages: **792**

Bibliografía