Tecnologías avanzadas para el desarrollo de Web Services

Parte 3

Agenda

- WS-Transaction
- WS-Business Activity
- WS-Policy
- Web Services REST

Transacciones

WS-Coordination

WS-AtomicTransaction

WS-BusinessActivity

WS-Transactions

- Compuesto por 3 estándares de la OASIS
  - WS – Coordination
  - WS – AtomicTransaction
  - WS – BusinessActivity

- Actualmente, se encuentran en la versión 1.2

WS-AtomicTransaction

Transacciones de corta duración

Transacciones

- Son un conjunto de tareas que cumplen las propiedades ACID

Transacciones

- Son un conjunto de tareas que cumplen las propiedades ACID
  - Atomicidad (Atomicity)
    - Se realizan todas las tareas o ninguna

Transacciones

- Son un conjunto de tareas que cumplen las propiedades ACID
  - Atomicidad (Atomicity)
    - Se realizan todas las tareas o ninguna
  - Consistencia (Consistency)
    - Las tareas realizadas no violan ninguna de las restricciones de integridad

Transacciones

- Son un conjunto de tareas que cumplen las propiedades ACID
  - Atomicidad (Atomicity)
    - Se realizan todas las tareas o ninguna
  - Consistencia (Consistency)
    - Las tareas realizadas no violan ninguna de las restricciones de integridad
  - Aislamiento (Isolation)
    - Las tareas no pueden acceder o ver datos que se encuentren en estados intermedios.

Transacciones

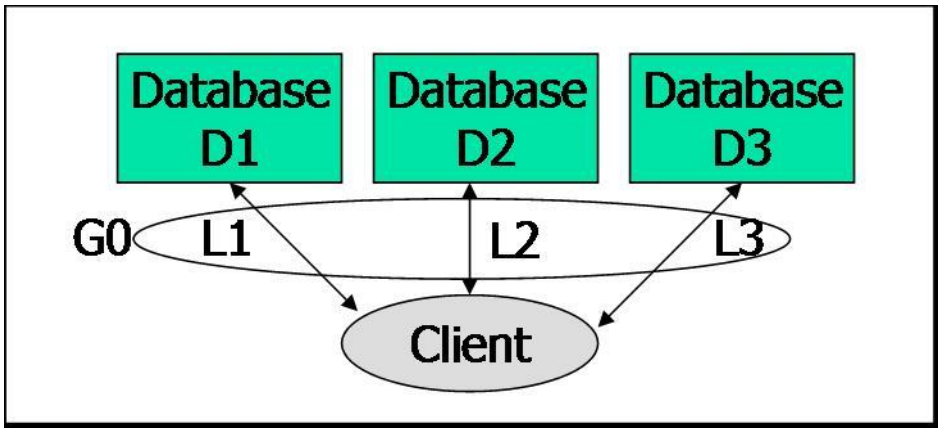
- Son un conjunto de tareas que cumplen las propiedades ACID
  - Atomicidad (Atomicity)
    - Se realizan todas las tareas o ninguna
  - Consistencia (Consistency)
    - Las tareas realizadas no violan ninguna de las restricciones de integridad
  - Aislamiento (Isolation)
    - Las tareas no pueden acceder o ver datos que se encuentren en estados intermedios.
  - Durabilidad (Durability)
    - Una vez realizada la transacción, se garantiza que esta persistirá a pesar de fallas en el sistema

Transacciones distribuidas

- Bases de datos independientes con sus propias transacciones locales (Li)

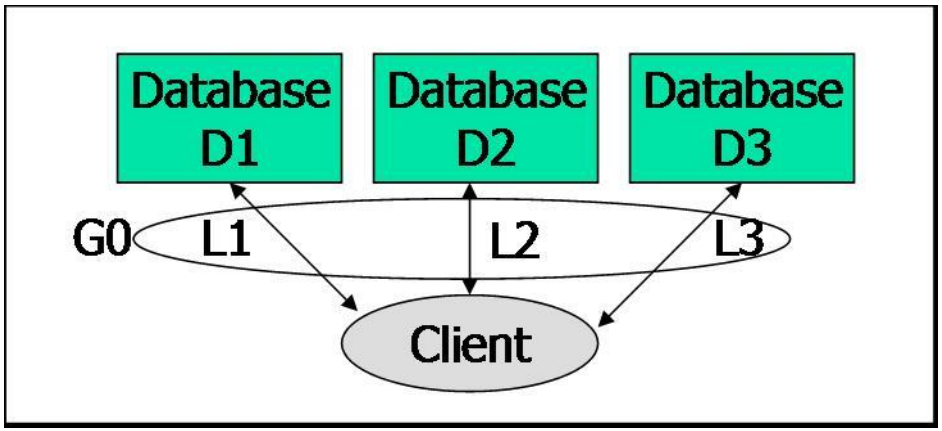
Transacciones distribuidas

- Bases de datos independientes con sus propias transacciones locales (Li)
- Una única transacción global (G0)



Transacciones distribuidas

- Bases de datos independientes con sus propias transacciones locales (Li)
- Una única transacción global (G0)



- Deseable mantener propiedades ACID!

Transacciones distribuidas

- Verificar propiedades ACID:
  - ACID(Li) => ACID(G0)

Transacciones distribuidas

- Verificar propiedades ACID:
  - ACID(Li) => ACID(G0)
    - *C(L1),C(L2),C(L3) => C(G0)*
      - *Si cada base de datos mantiene la consistencia, la transacción global es consistente*

Transacciones distribuidas

- Verificar propiedades ACID:
  - ACID(Li) => ACID(G0)
    - *C(L1),C(L2),C(L3) => C(G0)*
    - *I(L1),I(L2),I(L3) => I(G0)*
      - *Si cada base de datos mantiene los datos no visibles a otros actores, la transacción global cumple con la propiedad de aislación*

Transacciones distribuidas

- Verificar propiedades ACID:
  - ACID(Li) => ACID(G0)
    - *C(L1),C(L2),C(L3) => C(G0)*
    - *I(L1),I(L2),I(L3) => I(G0)*
    - *D(L1),D(L2),D(L3) => D(G0)*
      - *Si cada transacción persiste los datos y los mantiene a pesar de fallas en los sistemas, también lo hará la transacción global.*

Transacciones distribuidas

- Verificar propiedades ACID:
  - ACID(Li) => ACID(G0)
    - *C(L1),C(L2),C(L3) => C(G0)*
    - *I(L1),I(L2),I(L3) => I(G0)*
    - *D(L1),D(L2),D(L3) => D(G0)*

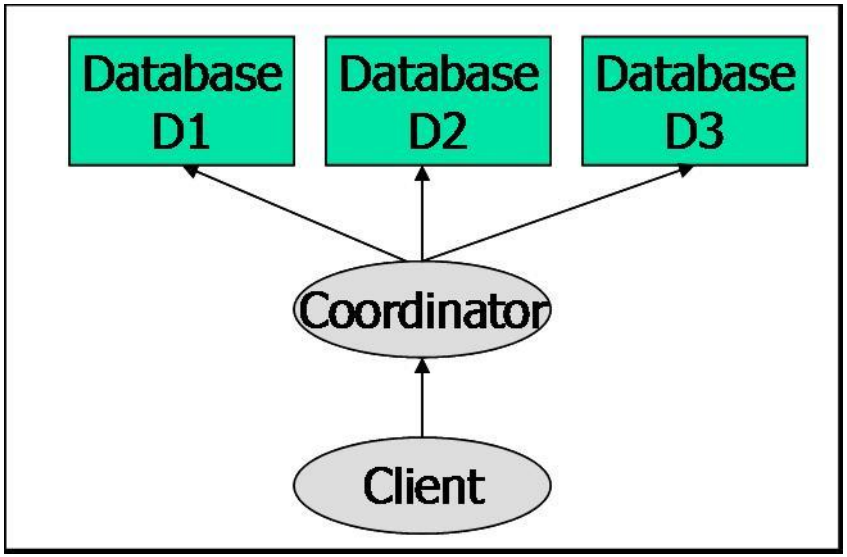
- La atomicidad es un problema!
  - *Abort(L1),Commit(L2),Commit(L3)*
    - *Localmente cada transacción mantiene la atomicidad pero la combinación no mantiene la atomicidad global*

Transacciones distribuidas

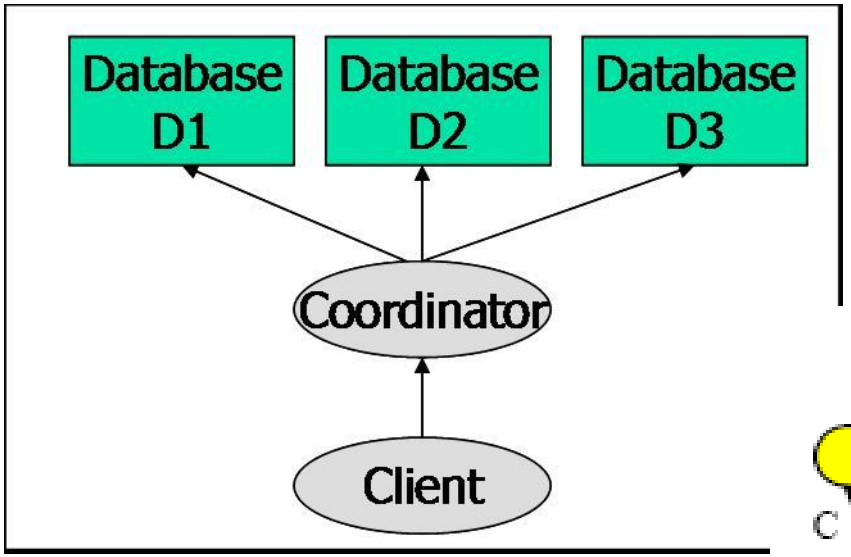
- Solución: agregar una operación *preparar* y un coordinador de la transacción global

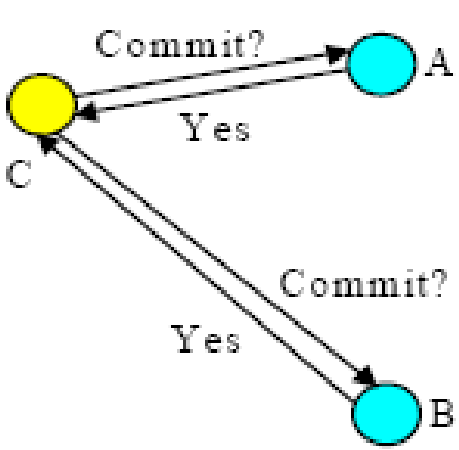
- El coordinador pregunta a cada participante
  - *Preparado para hacer commit?*
  - *Commit o Abort*
- Si alguno de los participantes aborta, la transacción global aborta
- A este protocolo se lo conoce como Two-Phase Commit

Two-Phase Commit (2PC)

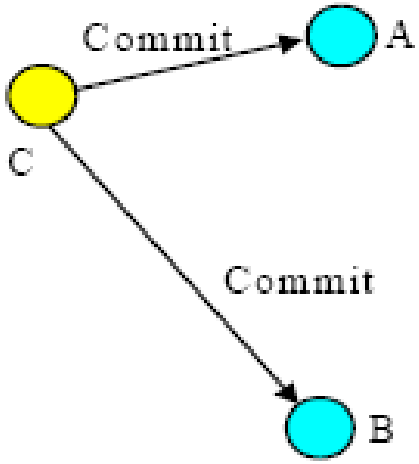


Two-Phase Commit (2PC)





Phase 1



Phase 2

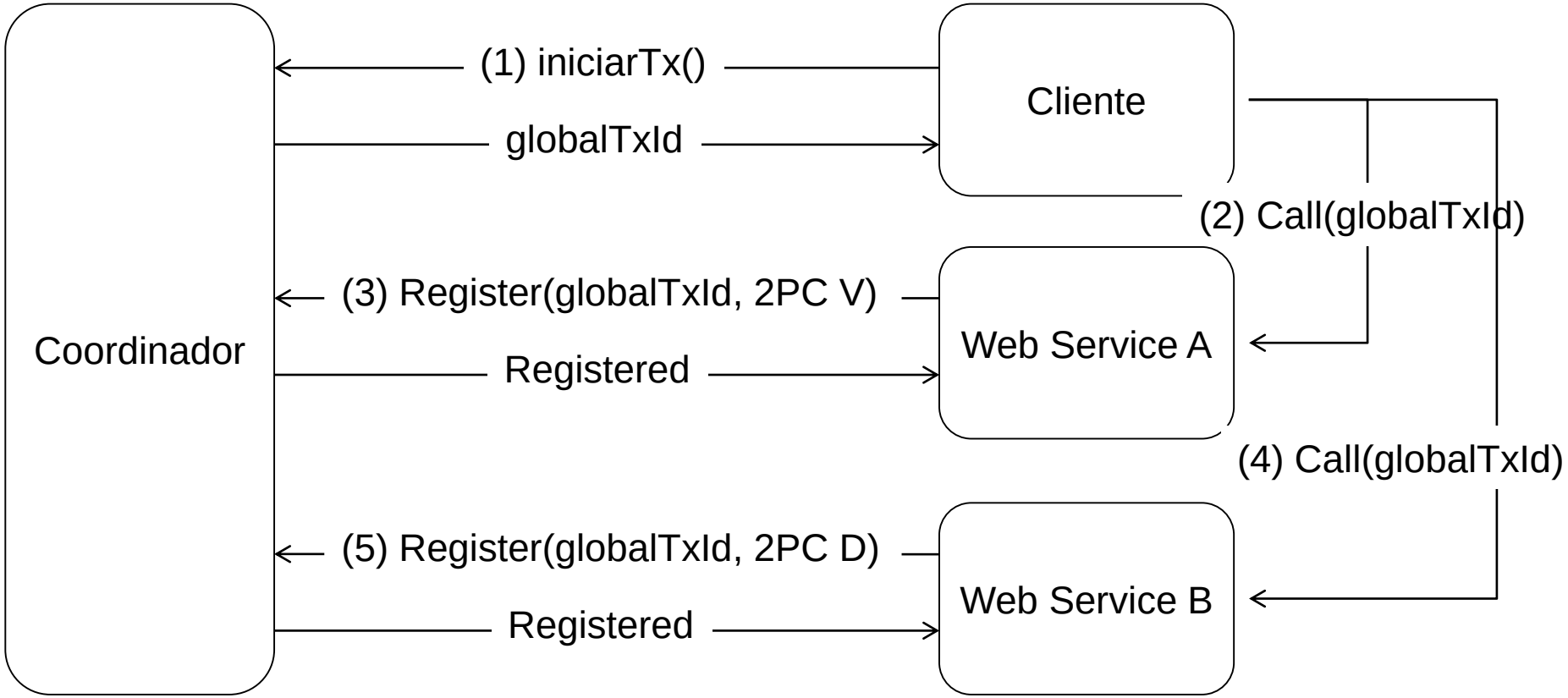
WS-AtomicTransaction

- WS-AT es un estándar de la OASIS con el propósito de proveer propiedades ACID a los Web Services.
- Protocolos
  - Durable Two-Phase Commit
  - Volatile Two-Phase Commit
  - Completition
- Actualmente, se encuentra en la versión 1.2

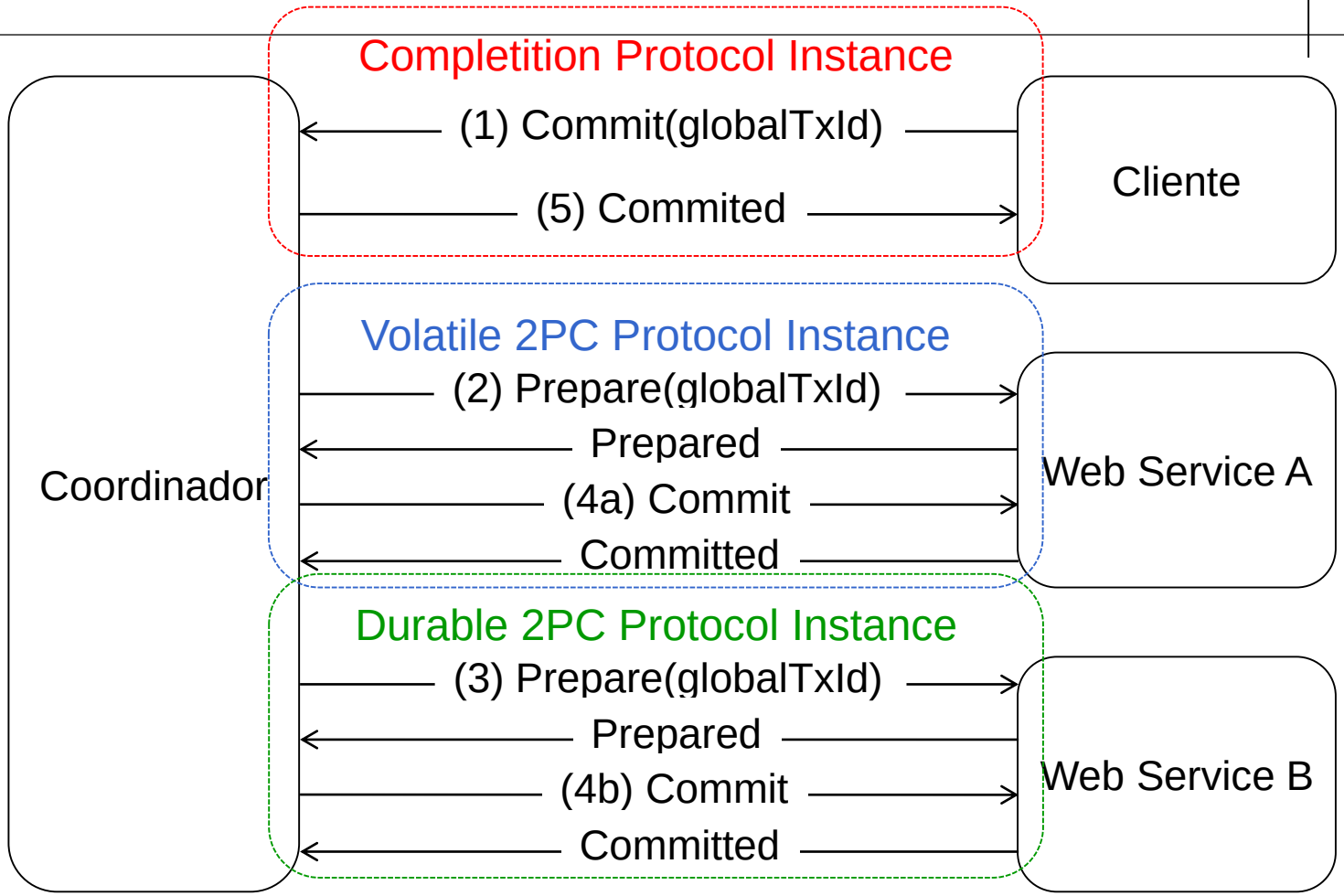
Protocolos

- Completition
  - Este protocolo es utilizado por los clientes con el fin de intentar confirmar o abortar la transacción
  - El resultado es la confirmación o cancelación de la transacción
- Volatile Two-Phase Commit
  - Aplica el protocolo 2PC a recursos volátiles que cambian frecuentemente (p. ej: cache, colas de mensajes en memoria)
- Durable Two-Phase Commit
  - Aplica el protocolo 2PC a recursos durables en el tiempo (p. ej: bases de datos, colas de mensajes persistentes)

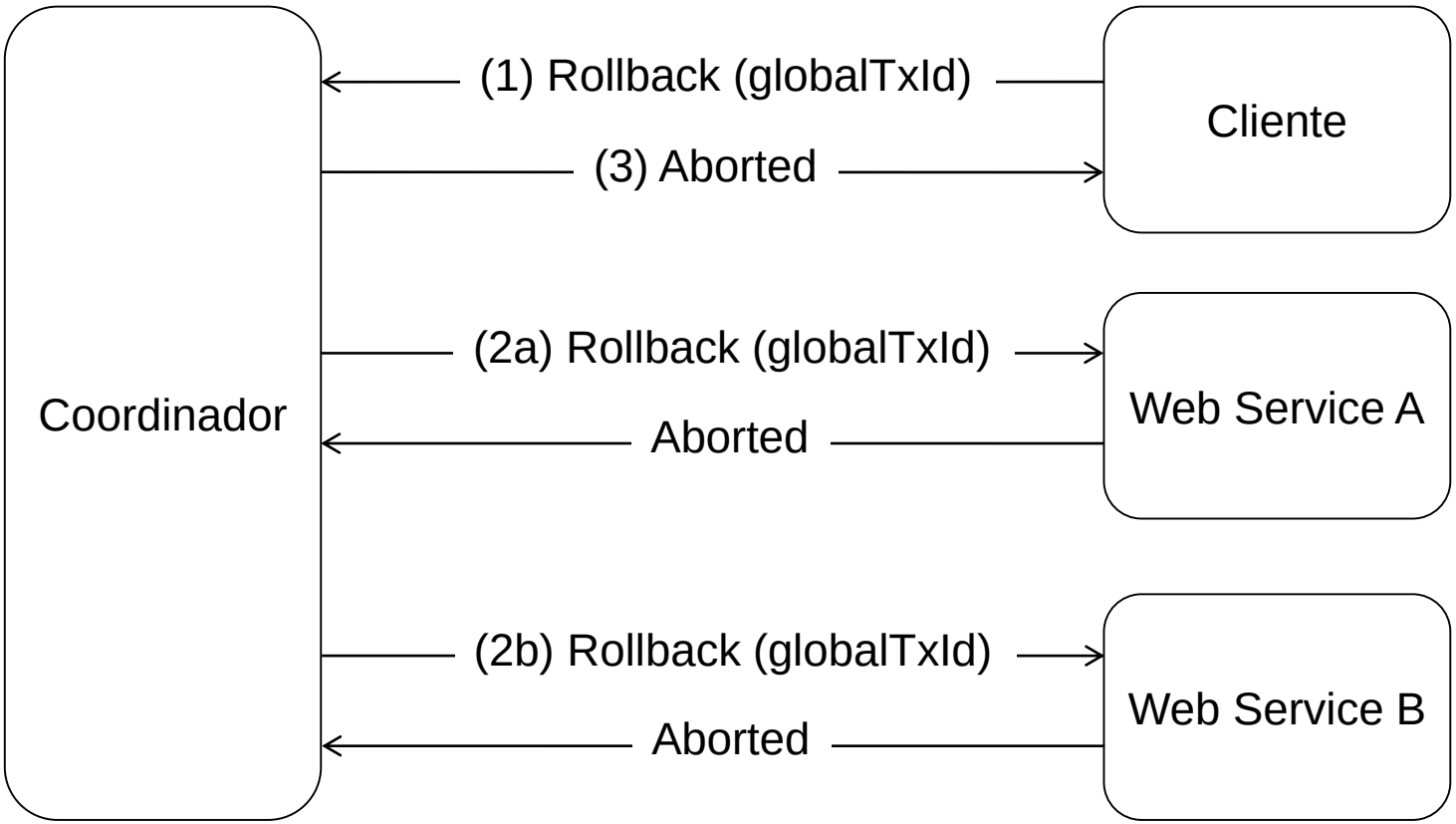
Atomic Transaction



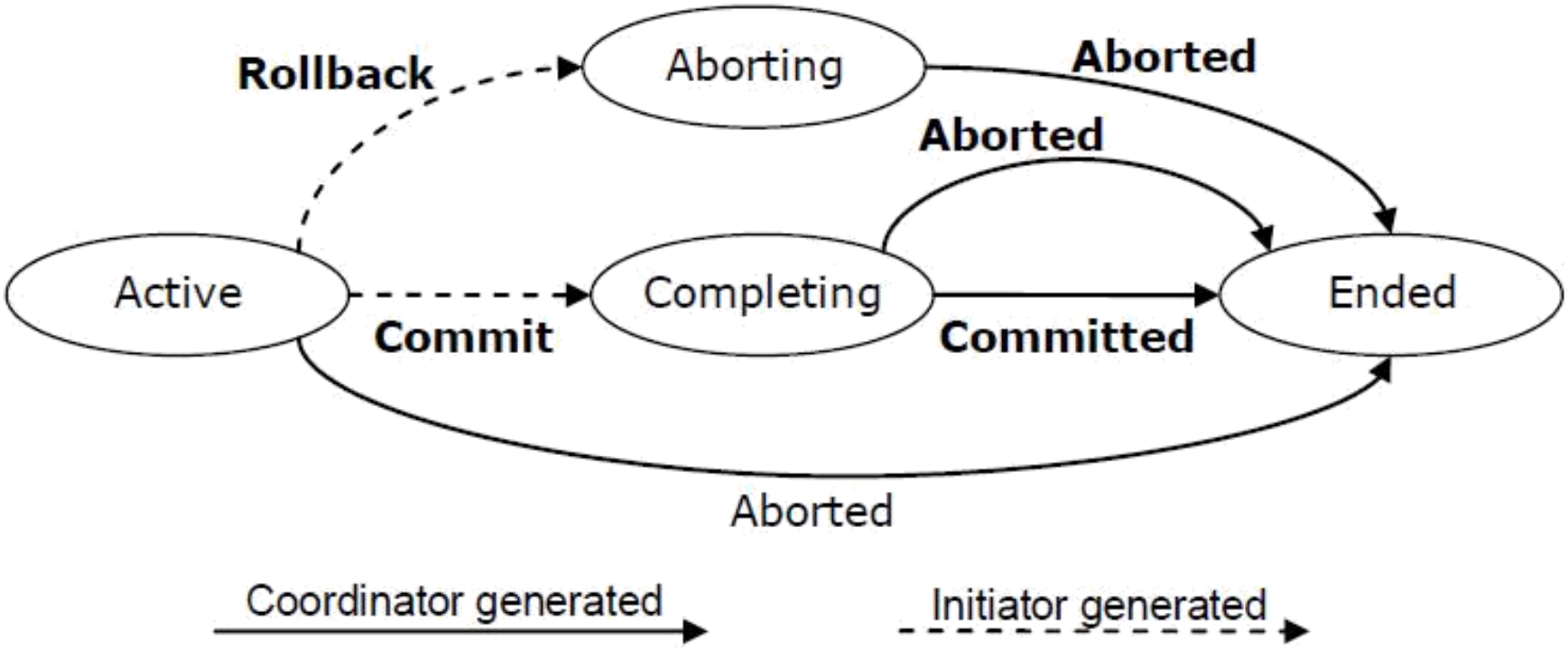
Atomic Transaction Commit



Atomic Transaction Abort

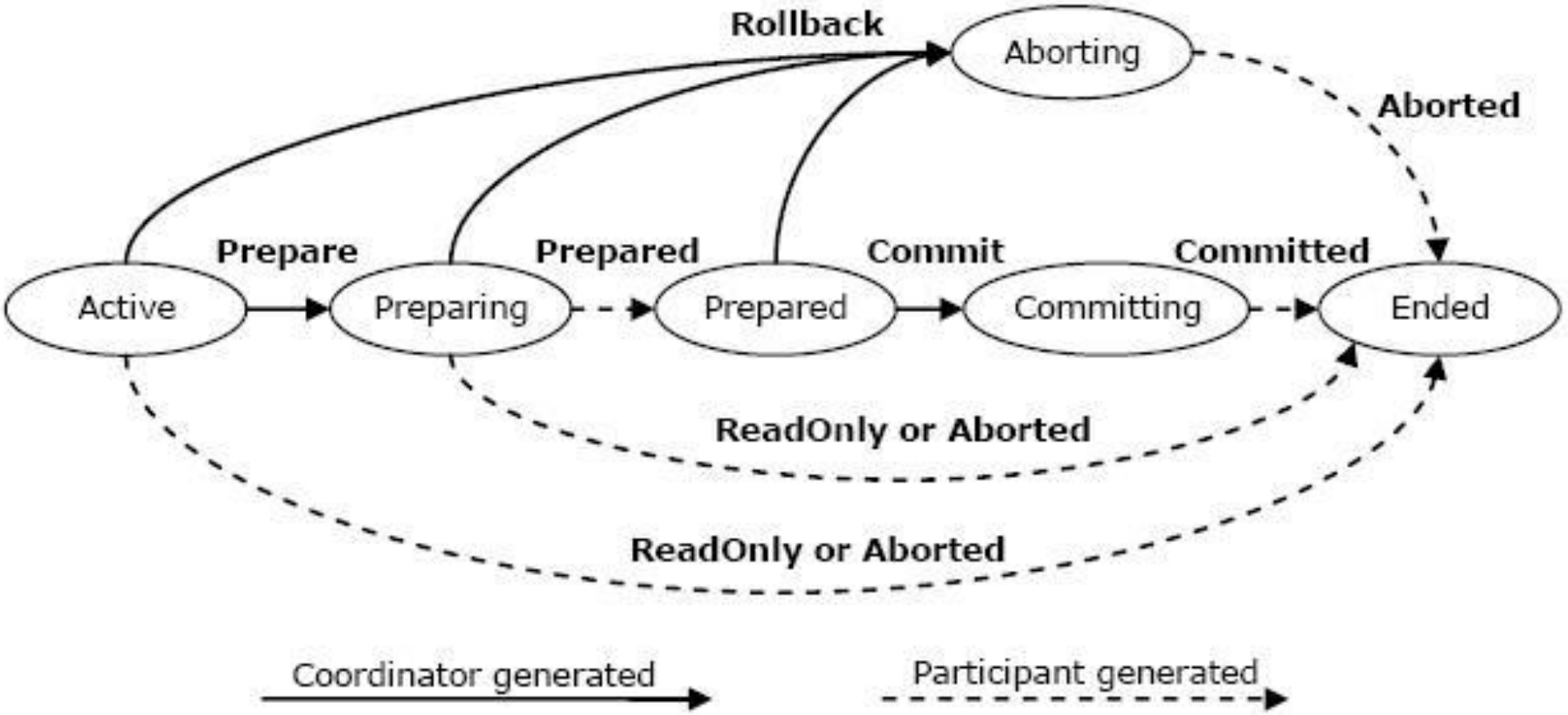


Protocolo Completition



- Estados y transiciones entre cliente y coordinador

Protocolo 2PC



- Estados y transiciones entre coordinador y participantes

Ejemplos SOAP

```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
 <env:Header>
 <wscoor:CoordinationContext
      env:mustUnderstand='1'
      xmlns:wscoor='http://docs.oasis-open.org/ws-tx/wscoor/2006/06'
      xmlns:ns3='http://www.w3.org/2005/08/addressing'>
 <wscoor:Identifier>urn:-3f570b7e:d61:4ad386e0:73</wscoor:Identifier>
 <wscoor:CoordinationType>http://docs.oasis-open.org/ws-tx/wsat/2006/06
 </wscoor:CoordinationType>
 <wscoor:RegistrationService>
 <ns3:Address>http://vmxp.localdomain:8080/ws-c11/RegistrationService
 </ns3:Address>
 <ns3:ReferenceParameters>
 <wsarj:InstanceIdentifier
 xmlns:wsarj='http://schemas.arjuna.com/ws/2005/10/wsarj'>-
3f570b7e:d61:4ad386e0:73</wsarj:InstanceIdentifier>
 </ns3:ReferenceParameters>
 <ns3:Metadata />
 </wscoor:RegistrationService>
 </wscoor:CoordinationContext>
  </env:Header>
 <env:Body>
   <ns1:depositar xmlns:ns1="http://services.lins.fing.edu
      <idCuenta>1</idCuenta>
      <monto>5000.0</monto>
   </ns1:depositar>
 </env:Body>
</env:Envelope>
```

globalTxId

Dirección del Coordinador

Comentarios

Comentarios

- WS-AT no es muy deseable para integración B2B
  - Alto acoplamiento entre sistemas
    - Locks en BDs
  - .NET requiere una comunicación segura vía SSL entre coordinadores para mejorar la seguridad
- WS-AT orientado a EAI
- En transacciones B2B es mejor usar WS-BA

WS-BusinessActivity

Transacciones de larga duración con Web Services

Transacciones

- Son un conjunto de tareas que cumplen las propiedades ACID
  - Atomicidad (Atomicity)
    - Se realizan todas las tareas o ninguna
  - Consistencia (Consistency)
    - Las tareas realizadas no violan ninguna de las restricciones de integridad
  - Aislamiento (Isolation)
    - Las tareas no pueden acceder o ver datos que se encuentren en estados intermedios.
  - Durabilidad (Durability)
    - Una vez realizada la transacción, se garantiza que esta persistirá a pesar de fallas en el sistema

Transacciones largas o Actividades de negocio

- Son un conjunto de tareas que cumplen las propiedades ACID
  - Atomicidad (Atomicity)
    - Se realizan todas las tareas o ninguna
  - Consistencia (Consistency)
    - Las tareas realizadas no violan ninguna de las restricciones de integridad
  - Aislamiento (Isolation)
    - Las tareas no pueden acceder o ver datos que se encuentren en estados intermedios.
  - Durabilidad (Durability)
    - Una vez realizada la transacción, se garantiza que esta persistirá a pesar de fallas en el sistema

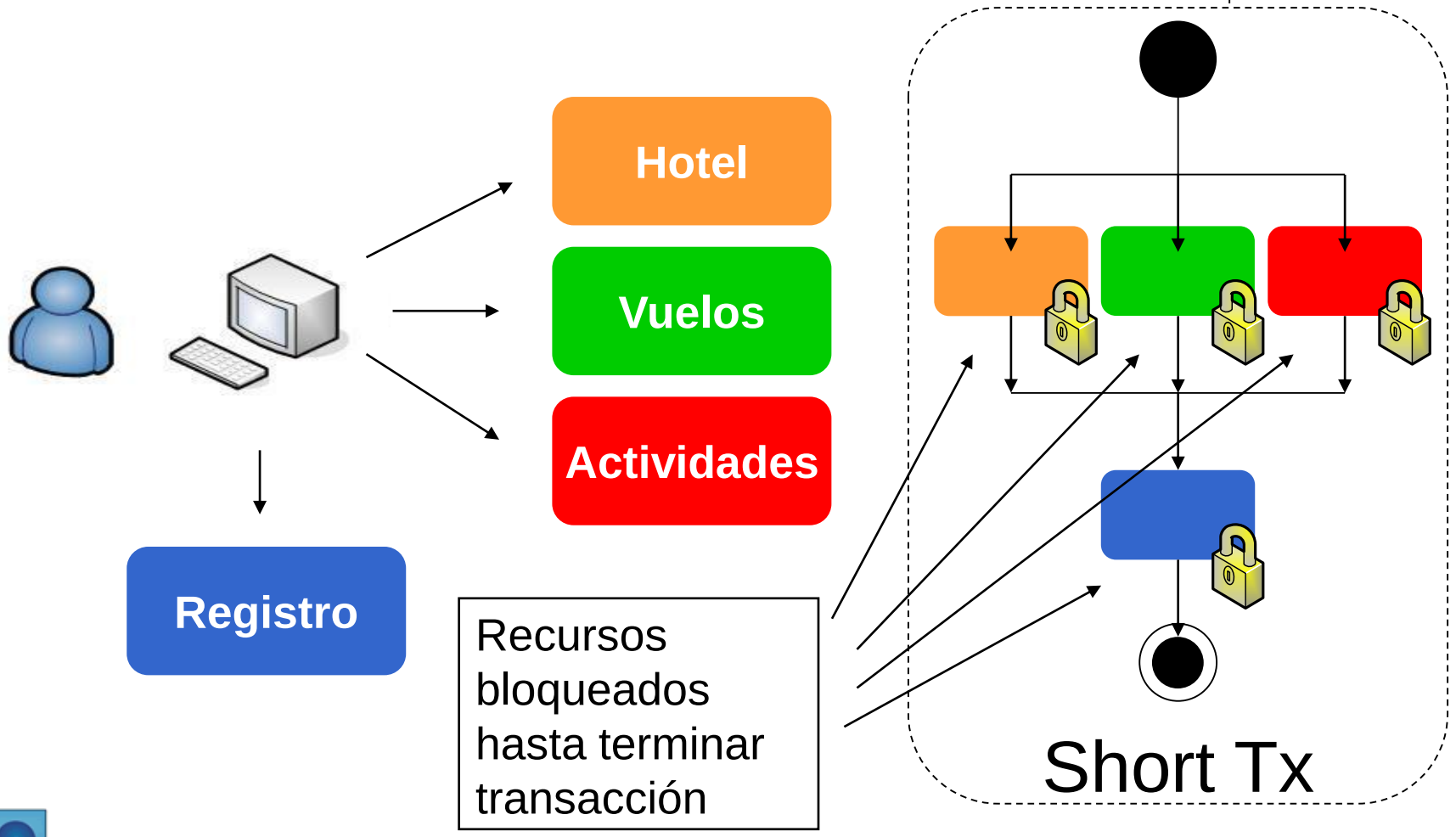
Actividades de negocio

- Pueden llegar a involucrar un gran número de transacciones atómicas independientes entre sí.
- Pueden llegar a consumir una gran cantidad de recursos y por un largo período de tiempo.
- Pueden llegar a demorar horas, días o incluso meses!
  - Aprobaciones, ensablado, manufactura o entregas pueden ser necesarias antes de dar una respuesta
- Las acciones intermedias son permanentes y pueden impactar por fuera del sistema.

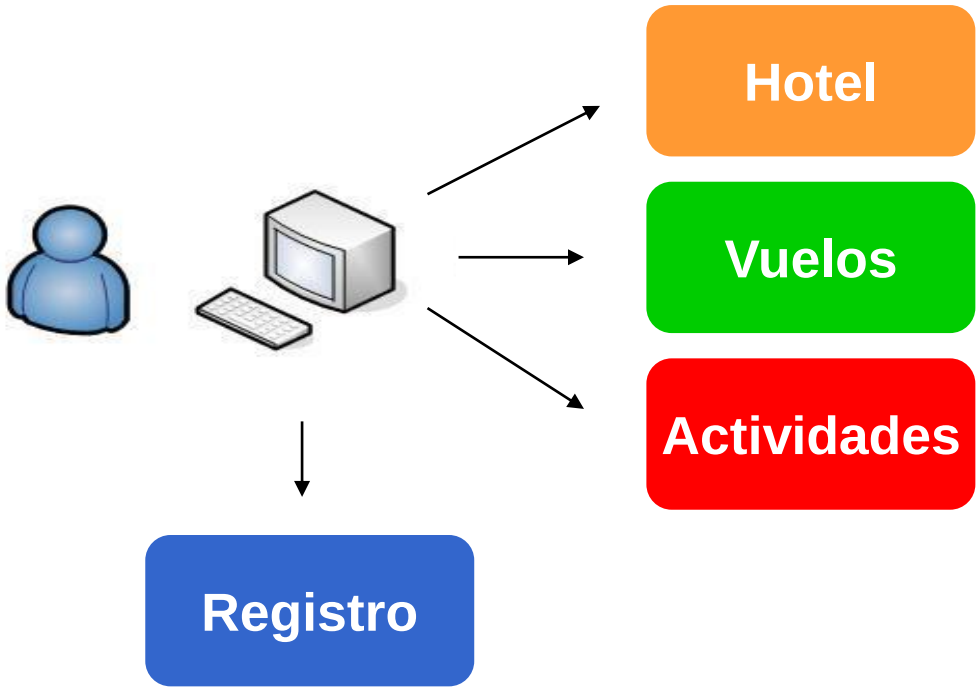
Actividades de negocio

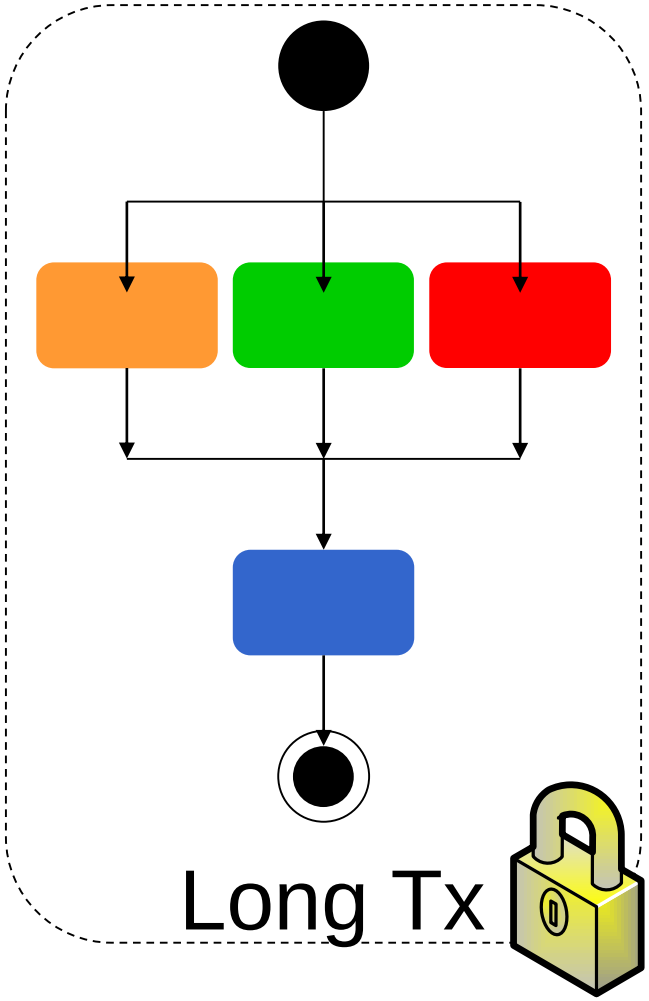
- No hay rollback como en las Transacciones tradicionales.
  - Abortar la transacción normalmente no es suficiente
- En caso de errores, se requiere de una actividad que deshaga las acciones tomadas.
  - El tratamiento de errores puede requerir cierta lógica de negocio
    - Tareas de compensación para revertir los efectos previamente confirmados por transacciones (atómicas) ya finalizadas.
- Sus participantes pueden pertenecer a diferentes dominios de confianza
  - Toda relación de confianza debe ser explicitamente definida.

Escenario


Hotel
Vuelos
Actividades
Registro
Recursos bloqueados hasta terminar transacción
Short Tx

Escenario





Escenario con transacciones de corta duración

- El usuario crea un paquete de viajes eligiendo un vuelo, un hotel y una serie de actividades.
  - El usuario puede realizar la selección en cualquier momento y tomarse el tiempo que desee.
  - Cuando finaliza, se registra el paquete en el sistema y se termina la transacción.

- Todas las tareas quedan enmarcadas en una única transacción corta, quedando todos los recursos bloqueados hasta que termine la transacción.
  - Un usuario bloquea a todos los demás!
  - Transacciones cortas no sirven debido a la naturaleza del problema.
  - Probemos otra cosa… Transacciones de larga duración!

Tx Largas+WS = WS-BA

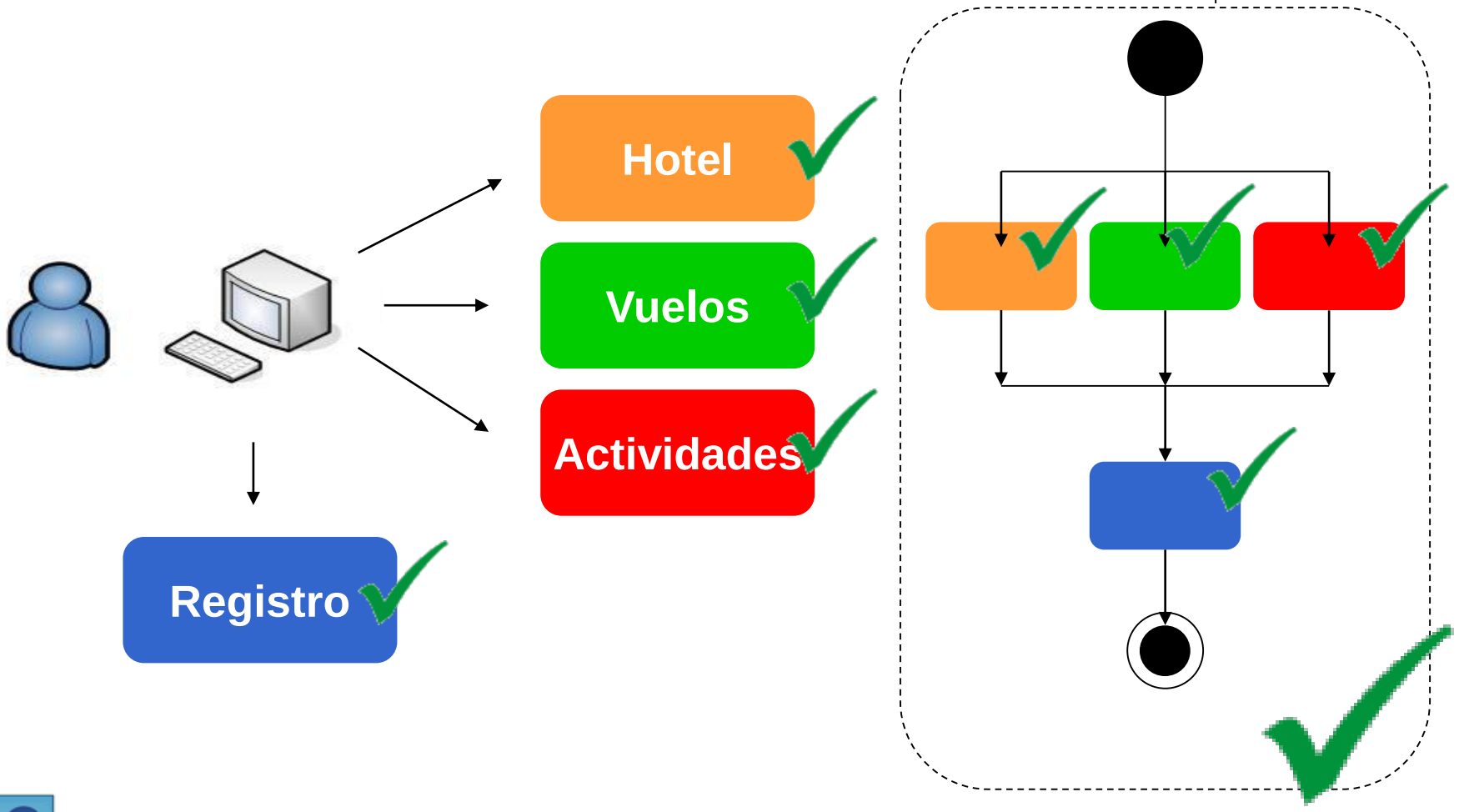
- WS-BA es un estándar de la OASIS con el propósito de definir mecanismos para la coordinación de actividades que involucren transacciones de larga duración y el tratamiento de errores en Web Services.

- Actualmente, en la versión 1.2

WS-BA: Contexto de ejecución

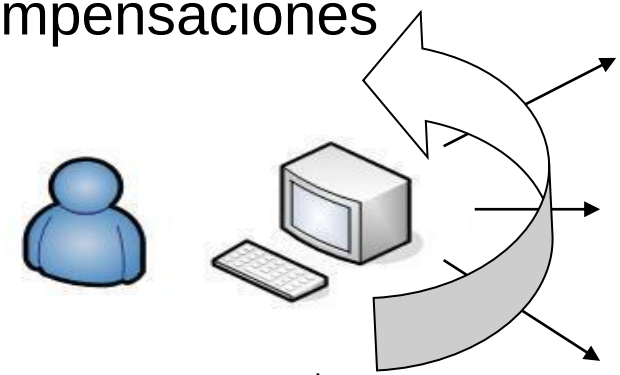
- Define dos contextos de ejecución
  - AtomicOutCome
    - Todos los participantes de la transacción llegan a un mismo y único resultado
    - Todos finalizan ok o todos compensan.
  - MixedOutCome
    - Permite finalizar una transacción donde los resultados de cada participantes sean independientes entre sí
      - Un participante: ok
      - Un participante: compensa
      - Transacción larga: ok
    - Opcional su implementación

Escenario: AtomicOutCome



Escenario: AtomicOutCome

Ejecución de compensaciones

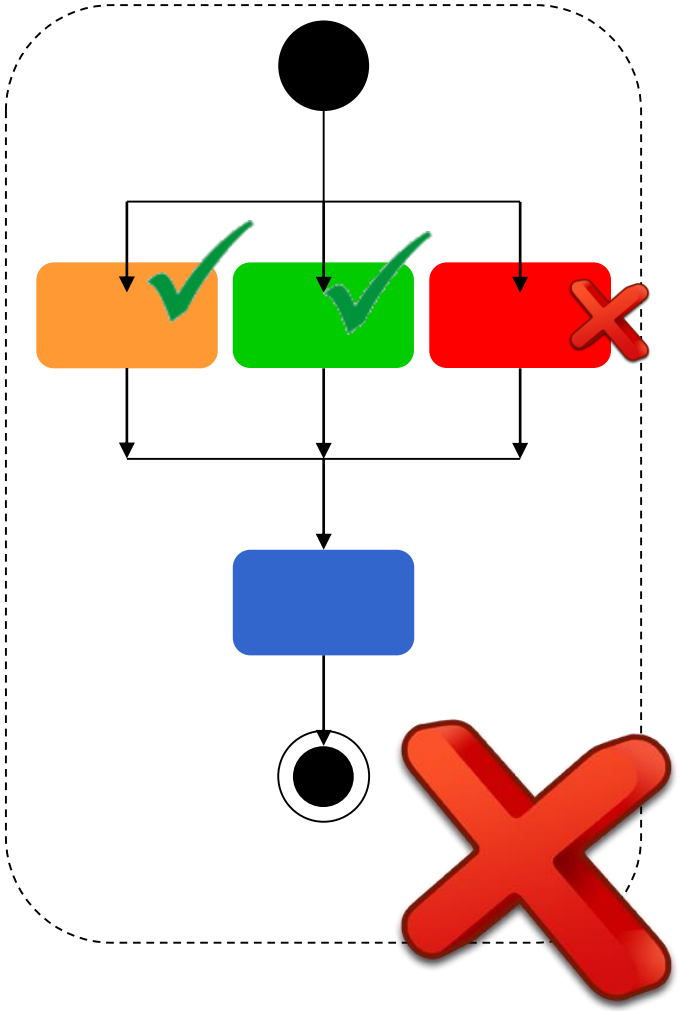


Registro

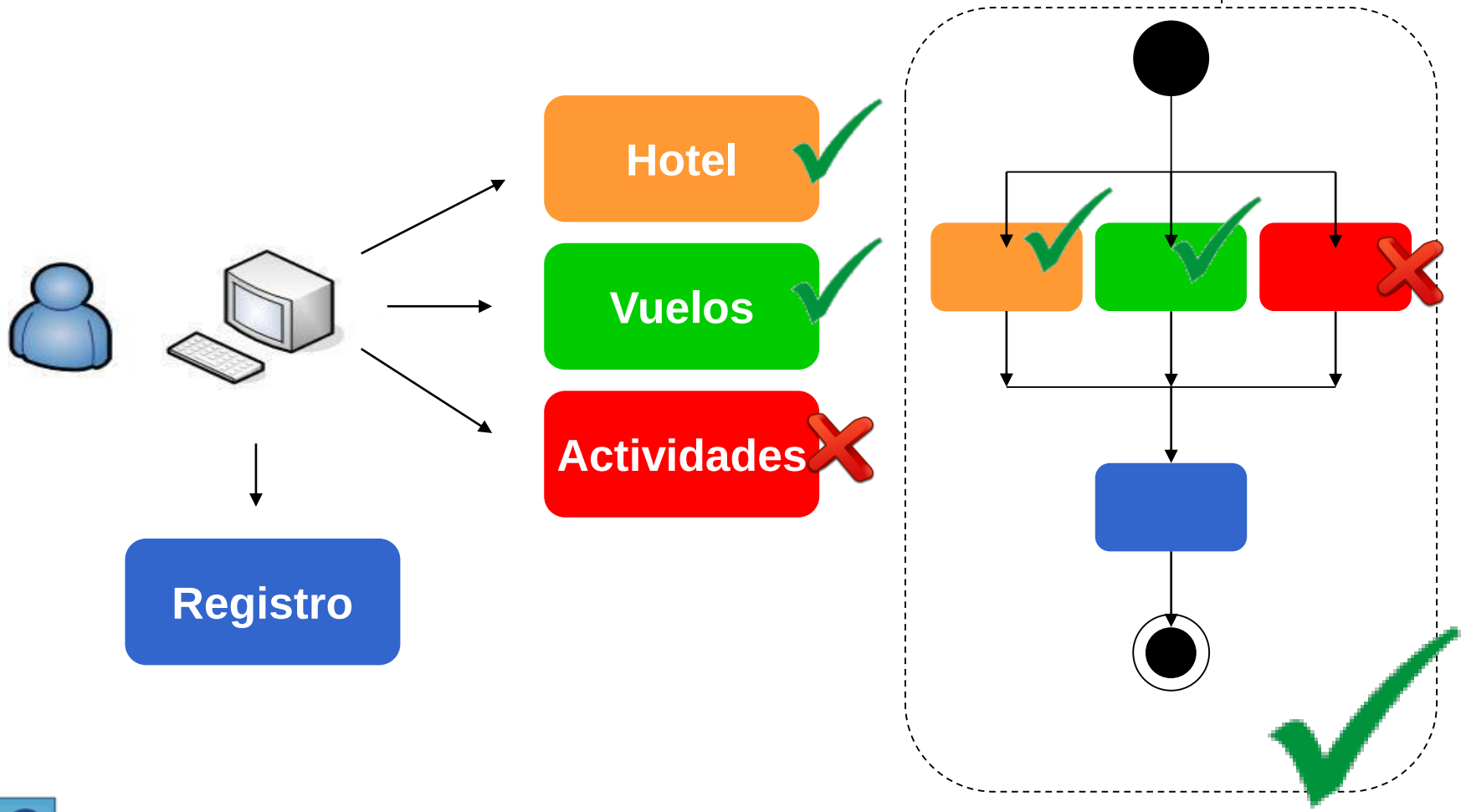
Hotel

Vuelos

Actividades



Escenario: MixedOutCome



Ejen

```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
 <env:Header>
  <wscoor:CoordinationContext env:mustUnderstand='1'
       xmlns:wscoor='http://schemas.xmlsoap.org/ws/2004/10/wscoor'>
   <wscoor:Identifier xmlns:wscoor='http://schemas.xmlsoap.org/ws/2004/10/wscoor'>
       urn:-3f57c379:570:4893ab85:cd
   </wscoor:Identifier>
   <wscoor:CoordinationType
       xmlns:wscoor='http://schemas.xmlsoap.org/ws/2004/10/wscoor'>
         http://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome
   </wscoor:CoordinationType>
   <wscoor:RegistrationService
       xmlns:wscoor='http://schemas.xmlsoap.org/ws/2004/10/wscoor'>
       <wsa:Address
xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'>
              http://localhost:8080/xts/soap/RegistrationCoordinator
       </wsa:Address>
       <wsa:ReferenceParameters
              xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'>
          <wsarj:InstanceIdentifier
                 xmlns:wsarj='http://schemas.arjuna.com/ws/2005/10/wsarj'>
                    -3f57c379:570:4893ab85:cd
          </wsarj:InstanceIdentifier>
       </wsa:ReferenceParameters>
   </wscoor:RegistrationService>
  </wscoor:CoordinationContext>
 </env:Header>
 <env:Body>
   <ns1:bookSeats xmlns:ns1='http://www.jboss.com/jbosstm/xts/demo/Re
     <how many>3</how many>
   </ns1:bookSeats>
 </env:Body>
</env:Envelope>
```

globalTxId

Contexto de ejecución

Dirección del Coordinador

Resumen

- WS-BA define mecanismos para la coordinación de actividades que involucren transacciones de larga duración y el tratamiento de errores.

- Define dos contextos de ejecución:
  - AtomicOutCome
  - MixedOutCome

Sin embargo…

- No tuvo suficiente apoyo de la industria
  - .NET nunca proveyó una implementación
- En la práctica Se utiliza un WS-BA “casero”
  - Implementación adhoc sin estandarización
- Surgieron otros enfoques para mantener la consistencia en sistemas distribuidos
  - Consistencia Eventual

Consistencia eventual

- Un enfoque menos estricto que las transacciones.
- Es posible continuar con el trabajo a pesar que el sistema no quede consistente inmediatamente.
- En un futuro, el sistema eventualmente quedará consistente.
- Durante este tiempo, el resto de los sistemas podrá acceder a estos datos “inconsistentes”.
- Favorece la disponibilidad frente la consistencia.

Consistencia eventual

- Si no hay nuevas actualizaciones sobre un objeto, eventualmente todos los accesos retornarán el último valor actualizado.

- Si no hay una falla, la ventana máxima de inconsistencia estará determinada por factores como retrasos en la comunicación, carga del sistema o el número de réplicas involucradas.

Consistencia eventual

- Consecuencia importante:
  - A nivel global, el sistema no siempre se encuentra en un estado consistente.
- Existe una “ventana de inconsistencia”
  - Período en el cual el sistema no es consistente
  - Puede causar que un determinado servicio/sistema trabaje con datos obsoletos hasta que esa “ventana” se acabe
- El desarrollador debe ser consciente se está trabajando con consistencia eventual

Escenario 1:
Domain Name System (DNS)

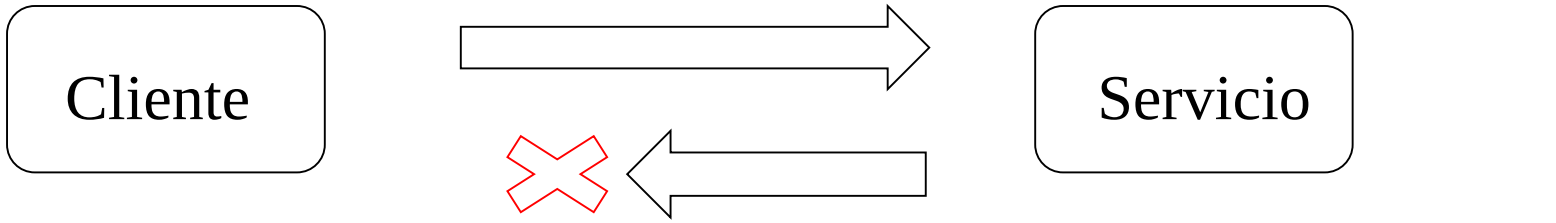
- Las actualizaciones de un nombre de host se distribuyen a lo largo de la red según un patrón de configuración y en combinación con políticas de caché

- Eventualmente todos los clientes van a acceder a los últimos datos actualizados del DNS.

Escenario 2: Pagos en línea

Laboratorio de Integracion de Sistemas

Fallos en la comunicación
p.ej: timeouts en la respuesta



Cliente no conoce resultado de la operación.
Debe cancelar solicitud

Servicio procesó correctamente el pago

Escenario 2: Pagos en línea

- Ante errores de comunicación (p.ej: timeouts) existen diferencias entre el resultado del cliente y del servicio.
  - Cliente: no se pudo procesar el pago
  - Servicio: el pago se procesó correctamente.
- El cliente debe comunicarse con el servicio para anular el pago realizado.
- Existe una ventana de tiempo donde el sistema global es inconsistente.

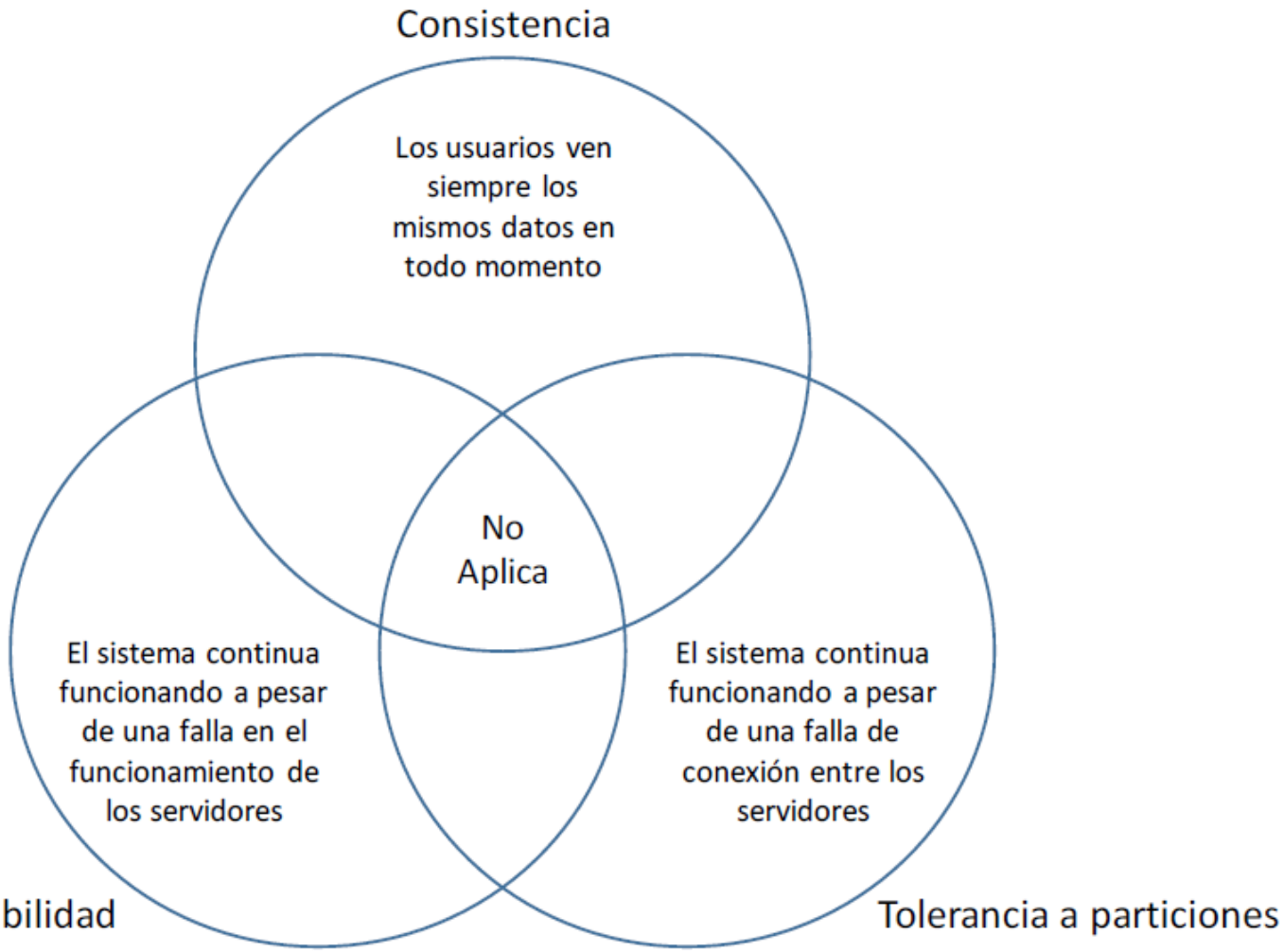
Teorema CAP

- En un sistema distribuido en el cual se comparten datos (una colección de nodos interconectados que comparten esos datos), solo se pueden obtener dos de las siguientes tres propiedades en un proceso de escritura/lectura de datos:
  - Consistencia
  - Disponibilidad
  - Tolerancia a fallos

Teorema CAP

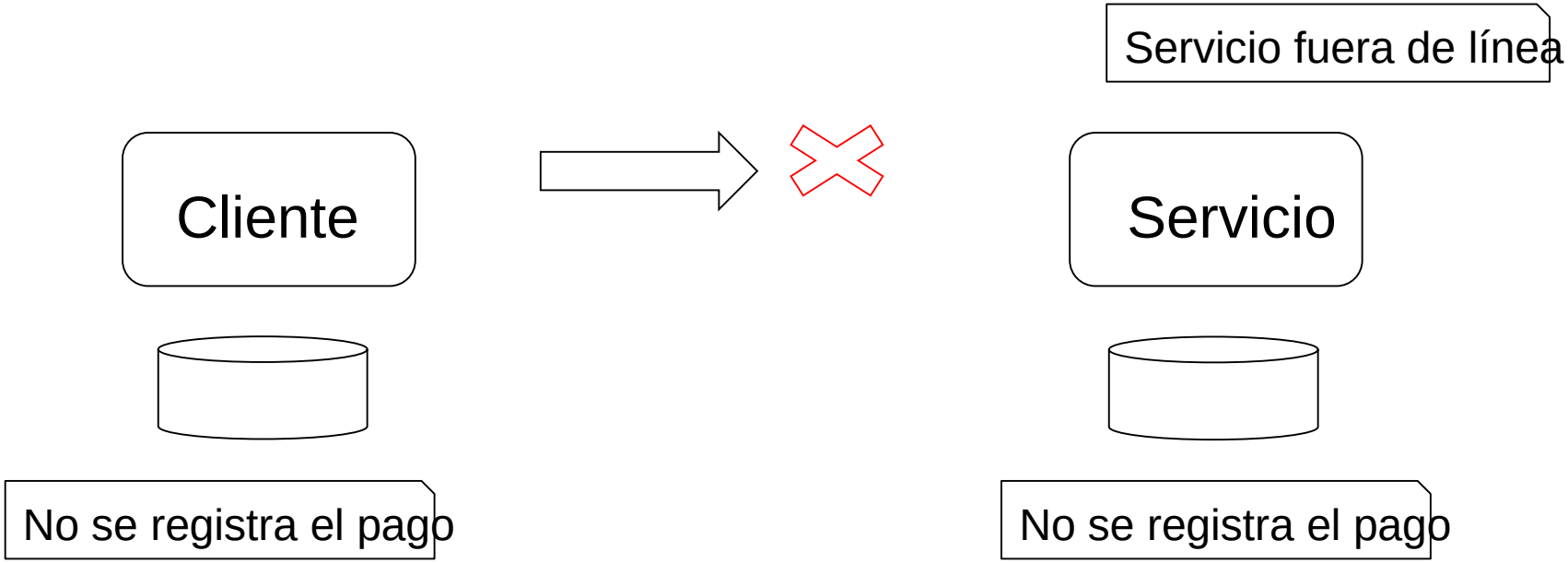
- Consistencia:
  - En todo nodo del sistema se puede observar la misma información
  - Se garantiza que una lectura de un dato en cualquier nodo, devolverá siempre el último valor escrito en el sistema global.
- Disponibilidad:
  - Cualquier solicitud a un nodo del sistema debe recibir una respuesta
  - Cualquier nodo que no haya fallado, retornará una respuesta en un período de tiempo razonable
- Tolerancia a particiones
  - El sistema debe seguir funcionando a pesar de fallas parciales en las comunicaciones.

Teorema CAP


Consistencia
Los usuarios ven siempre los mismos datos en todo momento
No Aplica
El sistema continua funcionando a pesar de una falla en el funcionamiento de los servidores
El sistema continua funcionando a pesar de una falla de conexión entre los servidores
Disponibilidad
Tolerancia a particiones

Escenario: Pagos en línea

- Consistencia + Tolerancia a fallos

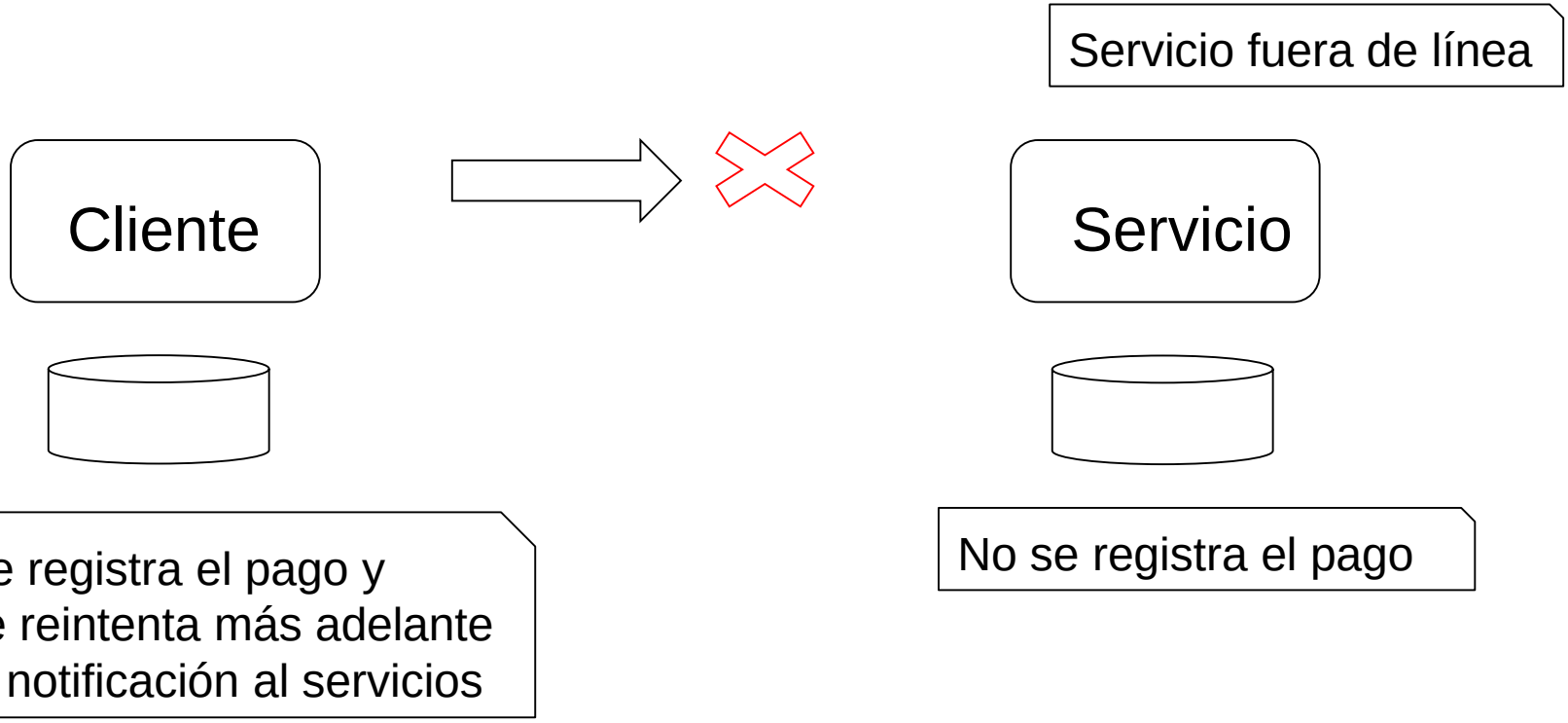


Escenario: Pagos en línea

- Consistencia + Tolerancia a fallos:
  - El servicio no está disponible
  - El cliente no puede comunicarse con el servicio para registrar el pago
  - Se prioriza la consistencia de la solución como un todo
  - Es preferible retornar que el cliente lance un error a dejar inconsistente el sistema
  - El cliente necesita la confirmación del servicio para responder correctamente.

Escenario: Pagos en línea

- Tolerancia a fallos + Disponibilidad



Escenario: Pagos en línea

- Tolerancia a fallos y disponibilidad:
  - El servicio no está disponible
  - El cliente no puede comunicarse con el servicio para registrar el pago
  - Se prioriza la disponibilidad de la solución como un todo.
  - El cliente no necesita la confirmación del servicio para responder correctamente.
  - Es preferible guardar el pago y reenviarlo más adelante a que el cliente lance un error.
  - Existe una ventana de tiempo donde el resultado global es inconsistente
  - Luego que el cliente se pueda comunicar con el servicio, el sistema volverá a ser consistente.

Escenario: Pagos en línea

- Consistencia + Disponibilidad

- No es posible este escenario en un sistema distribuido, ya que por su naturaleza, las comunicaciones fallan.
- Si tenemos este escenario, estamos contando con un único nodo en nuestra solución.
  - Se utilizan las técnicas tradicionales para mantener la consistencia.

Metadata

WS-Policy

WS-Policy

¿Qué hacen mis Web Services?

Introducción

- WS-Security
- WS-SecureConversation
- WS-Trust

Seguridad

- WS-ReliableMessaging
- WS-Addressing

Mensajería

- WS-AtomicTransaction
- WS-BusinessActivity

Transacciones

Introducción

- ¿Cómo se que un WS es seguro?
- ¿Cómo se que un WS es confiable?
- ¿Cómo se que un WS es transaccional?

- De alguna forma, tengo que poder describir las características o capacidades de los servicios
  - WS-Policy!

WS-Policy

- Define un modelo abstracto, independiente del dominio, que permite describir características, requerimientos y capacidades de un Web Service

- Delega a otras especificaciones la definición de políticas particulares a un dominio.
  - WS-SecurityPolicy
  - WS-ReliableMessagingPolicy

WS-Policy - Modelo

- Policy Assertions
  - Requerimiento o característica que describe al servicio.
- Policy Alternatives
  - Conjunto de assertions
- Policy
  - Conjunto de alternativas

WS-Policy – Ejemplo

```
<Policy>
    <ExactlyOne>
        <All>
            <SignedParts>
                <Body/>
            </SignedParts>
        </All>
        <All>
            <EncryptedParts>
                <Body/>
            </EncryptedParts>
        </All>
    </ExactlyOne>
    <ExactlyOne>
        <All>
            <IncludeTimestamp/>
        </All>
    </ExactlyOne>
</Policy>
```

Assertion

Assertion

Alternatives

Assertion

Alternatives

Policy

WS-SecurityPolicy

REpresentational State Transfer (REST)

Web Services livianos

Motivación

- WS-* es muy completo pero
  - Es complejo
    - Gran cantidad de estándares
    - Existen muchos más a los vistos en el curso
  - Es pesado
    - Recordar serialización SOAP

- Es necesario algo más simple y liviano
  - REST!

REST

- Representational State Transfer
- Es un estilo arquitectónico para aplicaciones que utilizan hipermedia interconectada
- Es aplicado para la construcción de servicios web, livianos, mantenibles y altamente escalables
- Un servicio basado en REST se denomina RESTfull
- Todo servicio RESTful utiliza HTTP/HTTPS como protocolo de transporte

Características principales

- Representaciones
- URIs
- Interfaz uniforme
- Falta de estado (Stateless)
- Uso de caches

Representaciones

- En REST todo es un recurso
  - Puede verse como un objeto en OOP
- Un recurso puede estar compuesto o relacionados a otros recursos
- Primero paso:
  - Identificar los recursos y sus relaciones
- Segundo paso:
  - Definir su o sus formas de representación.

Representaciones

- Ejemplos de recursos
  - La versión de un software
  - Una entrada en un blog
  - Información de una persona
  - Las ventas del primer trimestre del año
  - La relación entre dos personas
  - Una lista de bugs en nuestra aplicación

Representaciones

- Recurso: Persona
- Representación:
  - JSON: Deseable para que una página web realice invocaciones AJAX.
  - XML: Deseable para algunas comunicaciones B2B

```
{
    "Id": "1",
    "Name": "Pablo",
    "Email": "pablo@gmail.com",
    "Country": "Uruguay"
}
```

```
<Persona>
    <Id>1</Id>
    <Name>Pablo</Name>
    <Email>pablo@gmail.com</Email>
    <Country>Uruguay</Country>
</Persona>
```

Representaciones

- Una representación debe ser capaz de enlazar representación de recursos entre sí.
  - Se pueden usar referencias a otros recursos
  - HATEOAS:
    - Hipertext As The Engine Of Application State

```
<Persona>
    <Id>1</Id>
    <Name>Pablo</Name>
    <Email>Pablo@gmail.com</Email>
    <Link rel="address" href="http://services/address/4324" />
    <Country>Uruguay</Country>
</Persona>
```

Direccionamiento de recursos

- REST requiere que todo recurso tenga una identificación única
  - Una URI identifica un recurso
- La acción sobre el recurso se encuentra especificada por el verbo HTTP en la comunicación con el servicio Rest.
- Una URI NO debe decir nada sobre la acción a realizar
  - Esto permite invocar la misma URI con diferentes verbos HTTP para realizar diferentes acciones
- Por ejemplo, un recurso “persona” puede estar expuesto con esta URI http://myservice/personas/123

URIs

- Generalmente las URIs para acceder a recursos tienen el siguiente formato:
  - Protocol://ServiceName/ResourceType/ResourceID
- Algunas recomendaciones
  - Usar sustantivos en plural para nombrar un recurso
  - Evitar los espacios, usar "_" o "–"
  - Las URIs son sensibles al case.
- Evitar los verbos para describir la URI de un recurso
  - http://myservice/FetchPerson/1, http://myservice/DeletePerson/1
  - http://myservice/pagos/confirmar

URIs y query parameters

- Se utilizan query parameters para:
  - Filtros en búsquedas:
    - http://myservice/personas?name=juan
  - Ordenamiento de resultados
    - http://myservice/personas?sort=age
  - Personalizar la respuesta
    - http://myservice/personas/1?format=xml

- NO se deben utilizar para identificar recursos
  - http://myservice/personas?id=1

Interfaz uniforme

Método	Acción realizada en el servidor	Característica
GET	Lee un recurso	Segura
PUT	Inserta un nuevo recurso o lo actualiza si este ya existe	Idempotente
POST	Inserta un nuevo recurso	N/A
DELETE	Elimina un recurso	Idempotente
OPTIONS	Lista las acciones sobre un recurso	Segura
PATCH	Actualiza parcialmente un recurso	Puede ser idempotente

Seguridad e idempotencia

- Seguro
  - El método HTTP no modifica el estado del recurso
- Idempotente
  - No importa la cantidad de veces que se ejecute el método, el resultado siempre es el mismo

PUT vs POST

Solicitud	Acción
PUT http://myservice/personas	No funciona, PUT requiere una URI específica
PUT http://myservice/personas/123	Si no existe, inserta la persona con id=123. Si existe, la sobreescribe. Los datos de la persona van en el body http
POST http://myservcie/personas	Inserta una nueva persona cada vez que se realiza esta solicitud. Cada recurso persona se asocia a un nuevo Id definido por el servicio. La información de la persona va en el body http.

En ambos casos, es deseable retornar en la respuesta el recurso creado.

PATCH vs PUT

- En caso de existir el recurso, PUT hace un remplazo del recurso. El recurso previo se pierde
- PATCH solo actualiza los datos presentes en el body http. Realiza actualizaciones parciales

- En ambos casos es deseable retornar el recurso actualizado en la respuesta

Interfaz uniforme

- Los métodos indicados deben ser usados solo para el propósito con el que fueron definidos
- Nunca debería usarse un GET para crear un recurso o para eliminarlo
- Si bien HTTP provee una interfaz uniforme, queda en manos del desarrollador del servicio el uso que se les da
  - Ejemplos: PUT, PATCH, POST

Codigos HTTP

Codigo HTTP	
200 OK	Usado por GET, PUT, PATCH, DELETE en respuestas ok o POST en caso que no se haya creado el recurso
201 Created	Recurso creado en POST. Debe ir en conjunto un link (Location header) a la ubicación del nuevo recurso
204 No Content	Respuesta a una solicitud que no retorna un recurso (p.ej: DELETE)
400 Bad Request	Está mal formada la solicitud
401 Unauthorized	Cuando no se pudo autenticar al cliente
403 Forbidden	Cuando se autenticó correctamente al cliente pero no tiene permiso de acceso al recurso
404 Not Found	No se pudo encontrar el servicio
500 Internal Server Error	Error interno al procesar la solicitud

Escenario ejemplo

- Una empresa de venta de entradas desea implementar su servicio de venta mediante servicios REST. Mediante este servicio es posible consultar, pagar, anular e imprimir tickets de entrada.

Escenario ejemplo

Operación	VERBO + URI	Descripción
Consultar disponibilidad	GET http://myservice/tickets?evento=123&sort=precio	Se retornan los tickets disponibles para el evento, ordenados por precio
Comprar un ticket	POST http://myservice/ventas	En el body http se envía el recurso ticket a asociar a la venta
Anular un ticket	PATCH http://myservice/ventas/456	En el body http se envía el nuevo estado de la venta como anulado
Imprimir ticket	GET http://myservice/impresiones?tickets=265	Se retorna el recurso impresión del ticket con id 265

Falta de estado

- Un servicio RESTful no mantiene el estado de aplicación para ningún cliente
  - El servicio trata cada solicitud como una solicitud independiente
  - En ese sentido, las solicitudes que llegan al servicio no pueden estar correlacionadas por él
- Stateless
  - Request1: GET http://MyService/Persons/1 HTTP/1.1
  - Request2: GET http://MyService/Persons/2 HTTP/1.1
- Stateful
  - Request1: GET http://MyService/Persons/1 HTTP/1.1
  - Request2: GET http://MyService/NextPerson HTTP/1.1

Caching

- Caching es el concepto de almacenar los resultados generados para reutilizarlos, en lugar de generarlos en cada oportunidad
- Esto puede ser realizado en el cliente, en el servidor o en cualquier componente intermedio
- Si bien es una herramienta poderosa para mejorar la performance, si no se usa con cuidado, se le pueden enviar al cliente resultados obsoletos
- Podemos controlar el caching de los resultados a través de una serie de cabezales HTTP

Caching

Cabezal HTTP	Aplicación
Date	Fecha y hora en la cual el recurso fue creado
Last modified	Fecha y hora de última modificación del recurso en el servicio
Cache-control	Cabezal HTTP para controlar el uso del caché
Expires	Fecha de expiración del recurso. Para uso de clientes HTTP 1.0
Age	Duración en segundos desde que se tomó el recurso del servicio. Puede ser insertado por un intermediario.

Errores comunes

- Usar REST como un mecanismo para hacer invocaciones RPC
- Abusar del uso de POST
- Colocar acciones en la URI
- Enmascarar un servicio como un recurso
- Mantener sesiones en el servidor

Ejemplos

- Twitter
  - https://dev.twitter.com/rest/public
- Rotten Tomatoes
  - http://developer.rottentomatoes.com/docs

Ejemplos

- Estándares para WS Geográficos desarrollados enteramente en REST
  - Primeros estándares no disponibles para SOAP
  - http://www.opengeospatial.org/standards

REST vs SOAP

Característica	REST	SOAP
Sintáxis	XML, JSON (más popular)	XML
Formato	Ninguno definido	SOAP
Interfaz	WADL o RAML (ambos no estándar)	WSDL
Transporte	HTTP	Potencialmente varios HTTP más popular
Orientado a…	Recursos	Operaciones
Propósito	Escalabilidad y performance	Interoperabilidad
Operaciones	Limitadas	Ilimitadas
Seguridad	HTTPS, HTTP Basic Authentication, OAuth	WS-*
Otros requerimientos empresariales	N/A	WS-*

REST vs SOAP

- Ninguno es mejor que el otro a priori
- SOAP más apropiado para integración de sistemas heterogéneos con requerimientos empresariales
- REST está orientado a aplicaciones Web con gran cantidad de clientes y desconocidos
  - Escalar en clientes

Fin

LINS

Laboratorio de Integracion de Sistemas