6. Interoperabilidad

Con Liming Zhu

El pájaro temprano (A) llega y atrapa el gusano (B), tirando de la cuerda (C) y disparando la pistola (D). La bala (E) revienta el globo (F), dejando caer el ladrillo (G) en el bulbo (H) del atomizador (I) y disparando el perfume (J) en la esponja (K). A medida que la esponja gana peso, se baja y tira de la cuerda (L), levantando el extremo de la tabla (M). La bola de cañón (N) cae en la nariz del caballero durmiente. La cuerda atada a la bola de cañón libera el corcho (O) de la botella de vacío (P) y el agua helada cae sobre la cara del durmiente para ayudar a la bola de cañón en su buen trabajo.

—Rube Goldberg, instrucciones para “un simple despertador”

La interoperabilidad se refiere al grado en que dos o más sistemas pueden intercambiar información útil a través de interfaces en un contexto particular. La definición incluye no solo tener la capacidad de intercambiar datos (interoperabilidad sintáctica) sino también la capacidad de interpretar correctamente los datos que se intercambian (interoperabilidad semántica). Un sistema no puede ser interoperable de forma aislada. Cualquier discusión sobre la interoperabilidad de un sistema debe identificar con quién, con qué y bajo qué circunstancias, por lo tanto, la necesidad de incluir el contexto.

La interoperabilidad se ve afectada por los sistemas que se espera que interoperen. Si ya conocemos las interfaces de los sistemas externos con los que nuestro sistema interactuará, entonces podemos diseñar ese conocimiento en el sistema. O podemos diseñar nuestro sistema para que funcione de manera más genérica, de modo que la identidad y los servicios que proporciona otro sistema puedan vincularse más adelante en el ciclo de vida, en el momento de la compilación o en el tiempo de ejecución.

Como todos los atributos de calidad, la interoperabilidad no es una proposición de sí o no, sino que tiene matices de significado. Existen varios marcos de caracterización para la interoperabilidad, todos los cuales parecen definir cinco niveles de “madurez” de interoperabilidad (consulte la sección “ Para leer más ” al final de este capítulo para ver un indicador). El nivel más bajo significa sistemas que no comparten datos en absoluto, o no lo hacencon cualquier éxito. El nivel más alto

significa sistemas que trabajan juntos a la perfección, nunca cometen errores al interpretar las comunicaciones de los demás y comparten el mismo modelo semántico subyacente del mundo en el que trabajan.

"Intercambio de información a través de interfaces"

La interoperabilidad, como dijimos, se trata de dos o más sistemas que intercambian información a través de interfaces.

En este punto, necesitamos aclarar dos conceptos críticos centrales de esta discusión y enfatizar que estamos teniendo una visión amplia de cada uno.

El primero es lo que significa "intercambiar información". Esto puede significar algo tan simple como el programa A que llama al programa B con algunos parámetros. Sin embargo, dos sistemas (o partes de un sistema) pueden intercambiar información incluso si nunca se comunican directamente entre sí. ¿Alguna vez tuviste una conversación como la siguiente en la escuela secundaria? "Charlene dijo que Kim le dijo que Trevor escuchó que Heather quiere venir a tu fiesta". Por supuesto, el protocolo de la escuela secundaria evitaría la posibilidad de responder directamente a Heather. En su lugar, su respuesta (si le gusta a Heather) podría ser, "Genial", que se abrirá camino a través de Charlene, Kim y Trevor. Tú y Heather intercambiaron información, pero nunca hablaron el uno con el otro. (Esperamos que tengan que hablar entre ellos en la fiesta).

Las entidades pueden intercambiar información de manera aún menos directa. Si tengo una idea del comportamiento de un programa, y diseño mi programa para que funcione con ese comportamiento, los dos programas también intercambiaron información, pero no en tiempo de ejecución.

Uno de los desastres de software más infames en la historia ocurrió cuando un sistema antimisiles no pudo interceptar un cohete balístico entrante en la Operación Tormenta del Desierto en 1991, lo que resultó en 28 muertes. Uno de los componentes de software del misil "espera" que se apague y reinicie periódicamente, para que pueda recalibrar su marco de orientación desde un punto inicial conocido. El software había estado funcionando durante unas 100 horas cuando se lanzó el misil, y los errores de cálculo se habían acumulado hasta el punto en que la idea del componente del software de su orientación se había alejado de la verdad.

Los sistemas (o componentes dentro de los sistemas) a menudo tienen o incorporan expectativas sobre los comportamientos de sus socios de "intercambio de información". El supuesto de que todo interactuaba con el componente errante en el ejemplo anterior era que su precisión no *se* degradaba con el tiempo. El resultado fue un sistema de partes que no funcionaron juntas correctamente para resolver el problema que debían.

El segundo concepto que debemos enfatizar es lo que entendemos por "interfaz". Una vez más, queremos decir algo más allá del caso simple: una descripción sintáctica de los programas de un componente y el tipo y número de

sus parámetros, más comúnmente realizados como API. Eso es necesario parainteroperabilidad: heck, es necesario si desea que su software se compile correctamente, pero no es suficiente. Para ilustrar este concepto, usaremos otra analogía de "conversación". ¿Alguna vez su pareja o cónyuge ha vuelto a casa, ha cerrado la puerta y, cuando pregunta qué pasa, responde “¡Nada!”? Si es así, entonces debería ser capaz de apreciar la gran diferencia entre la sintaxis y la semántica y el papel de las expectativas en la comprensión de cómo se comporta una entidad. Como queremos sistemas y componentes interoperables, y no simplemente los que se compilan juntos, necesitamos una barra más alta para las interfaces que solo una declaración de sintaxis. Por "interfaz", nos referimos al conjunto de suposiciones que puede hacer con seguridad sobre una entidad. Por ejemplo, es una suposición segura de que todo lo que esté mal con su cónyuge / pareja no es "Nada,*mucho* más allá de las palabras que dicen. Y también es una suposición segura de que nada sobre la degradación de la precisión de nuestro componente de misiles a lo largo del tiempo se encontraba en su API, y sin embargo, eso era una parte crítica de su interfaz.

- *PCC*

Estas son algunas de las razones por las que podría querer que los sistemas interoperen:

• Su sistema proporciona un servicio para ser utilizado por una colección de sistemas desconocidos. Estos sistemas necesitan interoperar con su sistema aunque no sepa nada sobre ellos. Un ejemplo es un servicio como Google Maps.

• Estás construyendo capacidades a partir de sistemas existentes. Por ejemplo, uno de los sistemas existentes es responsable de detectar su entorno, otro es responsable de procesar los datos sin procesar, un tercero es responsable de interpretar los datos y el último es responsable de producir y distribuir una representación de lo que se detectó. . Un ejemplo es un sistema de detección de tráfico donde la entrada proviene de vehículos individuales, los datos sin procesar se procesan en unidades de medida comunes, se interpretan y fusionan, y se transmite información sobre la congestión del tráfico.

Estos ejemplos resaltan dos aspectos importantes de la interoperabilidad:

1. *Descubrimiento*. El consumidor de un servicio debe descubrir (posiblemente en tiempo de ejecución, posiblemente antes del tiempo de ejecución) la ubicación, la identidad y la interfaz del servicio.

2. *Manejo de la respuesta*. Hay tres posibilidades distintas:

• El servicio informa al solicitante con la respuesta.

• El servicio envía su respuesta a otro sistema.

• El servicio transmite su respuesta a cualquier parte interesada.

Estos elementos, el descubrimiento y la disposición de la respuesta, junto con la administración de interfaces, gobiernan nuestra discusión de escenarios y tácticas para la interoperabilidad.

Sistemas de sistemas

Si tiene un grupo de sistemas que están interoperando para lograr un propósito conjunto, tiene lo que se llama un *sistema de sistemas* (SoS). Un SoS es una disposición de sistemas que resulta cuando los sistemas independientes y útiles se integran en un sistema más grande que ofrece capacidades únicas. La tabla 6.1 muestra una categorización de SoSs.

Tabla 6.1. Taxonomía de los sistemas de sistemas *

Directed	SoS objectives, centralized management, funding, and authority for the overall SoS are in place. Systems are subordinated to the SoS.
Acknowledged	SoS objectives, centralized management, funding, and authority in place. However, systems retain their own management, funding, and authority in parallel with the SoS.
Collaborative	There are no overall objectives, centralized management, authority, responsibility, or funding at the SoS level. Systems voluntarily work together to address shared or common interests.
Virtual	Like collaborative, but systems don't know about each other.

* *La taxonomía mostrada es una extensión del trabajo realizado por Mark Maier en 1998.*

En los SoS dirigidos y reconocidos, hay un intento deliberado de crear un SoS. La diferencia clave es que en el primero, existe una gestión de nivel SoS que ejerce control sobre los sistemas constituyentes, mientras que en el segundo, los sistemas constituyentes conservan un alto grado de autonomía en su propia evolución. Los sistemas de sistemas colaborativos y virtuales son más ad hoc, en ausencia de una autoridad o fuente de financiamiento global y, en el caso de un SoS virtual, incluso sin el conocimiento sobre el alcance y la membresía del SoS.

El caso colaborativo es bastante común. Considere el ejemplo de Google Maps de la introducción. Google es el administrador y la autoridad de financiamiento para el servicio de mapas. Cada uso de los mapas en una aplicación (un SoS) tiene su propia administración y autoridad de financiamiento, y no hay una administración general de todas las aplicaciones que usan Google Maps. Las

diversas organizaciones involucradas en las aplicaciones colaboran (de manera explícita o implícita) para permitir que las aplicaciones funcionen correctamente.

Un SoS virtual involucra sistemas grandes y es mucho más ad hoc. Por ejemplo, hay más de 3,000 compañías eléctricas en la red eléctrica de los EE. UU., Cada estado tiene una comisión de servicios públicos que supervisa a las empresas de servicios públicos que operan en su estado, y el Departamento de Energía federal proporciona algún nivel de orientación de políticas. Muchos de los sistemas dentro de la red eléctrica deben interoperar, pero no hay autoridad de gestión para el sistema en general.

6.1. ESCENARIO DE INTEROPERABILIDAD GENERAL

Las siguientes son las partes de un escenario general de interoperabilidad:

• *Fuente de estímulo* . Un sistema inicia una solicitud para interoperar con otro sistema.

• *Estímulo* . Una solicitud para intercambiar información entre sistemas.

• *Artefactos* . Los sistemas que deseen interoperar.

• *Medio ambiente* . Los sistemas que desean interoperar se descubren en tiempo de ejecución o se conocen antes del tiempo de ejecución.

• *Respuesta* . La solicitud de interoperar resulta en el intercambio de información. La información es entendida por la parte receptora de manera sintáctica y semántica. Alternativamente, la solicitud es rechazada y las entidades apropiadas son notificadas. En cualquier caso, la solicitud puede ser registrada.

• *Medida de respuesta* . El porcentaje de intercambios de información procesados correctamente o el porcentaje de intercambios de información rechazados correctamente.

La Figura 6.1 da un ejemplo: nuestro sistema de información del vehículo envía nuestra ubicación actual al sistema de monitoreo de tráfico. El sistema de monitoreo de tráfico combina nuestra ubicación con otra información, superpone esta información en un mapa de Google y la difunde. Nuestra información de ubicación está correctamente incluida con una probabilidad del 99.9%.

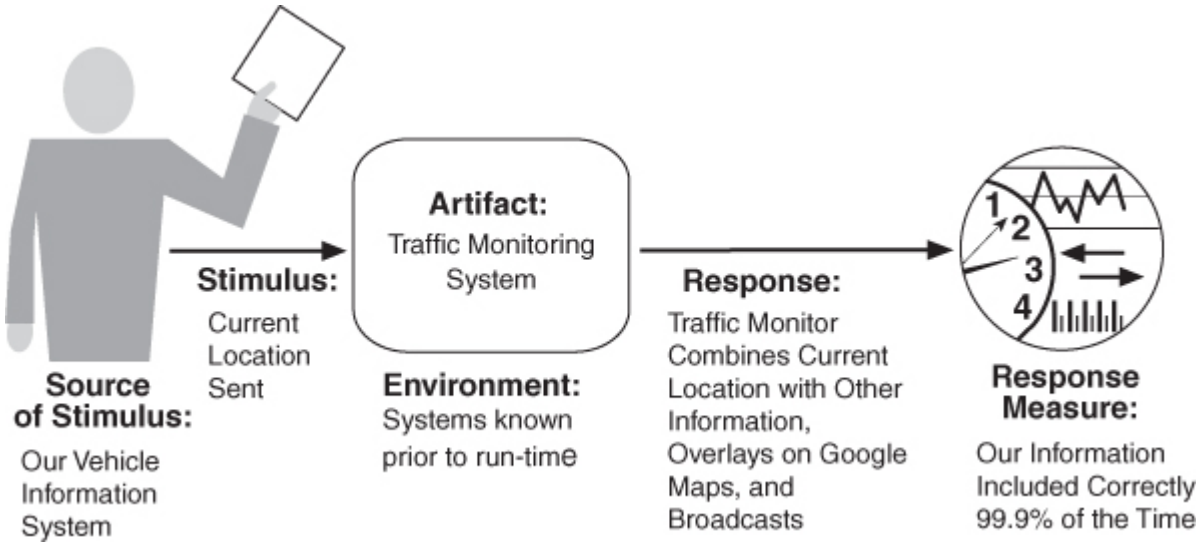

Figura 6.1. Escenario de interoperabilidad concreto de muestra.

La Tabla 6.2 presenta los valores posibles para cada parte de un escenario de interoperabilidad.

Tabla 6.2. Escenario de Interoperabilidad General

Portion of Scenario	Possible Values
Source	A system initiates a request to interoperate with another system.
Stimulus	A request to exchange information among system(s).
Artifact	The systems that wish to interoperate.
Environment	System(s) wishing to interoperate are discovered at runtime or known prior to runtime.
Response	One or more of the following: ▪ The request is (appropriately) rejected and appropriate entities (people or systems) are notified. ▪ The request is (appropriately) accepted and information is exchanged successfully. ▪ The request is logged by one or more of the involved systems.
Response Measure	One or more of the following: ▪ Percentage of information exchanges correctly processed ▪ Percentage of information exchanges correctly rejected

Jabón vs. Descanso

Si desea permitir que las aplicaciones basadas en la web interoperen, tiene dos opciones principales de tecnología disponibles en la actualidad: (1) WS * y

SOAP (que una vez significaba "Protocolo simple de acceso a objetos", pero ese acrónimo ya no es bendecido) y (2) REST (que significa "Transferencia de Estado de Representación" y, por lo tanto, a veces se escribe ReST). ¿Cómo podemos comparar estas tecnologías? ¿Para qué sirve cada uno? ¿Cuáles son los peligros de la carretera que debe tener en cuenta? Esto es un poco de una comparación de manzanas y naranjas, pero intentaré dibujar el paisaje.

SOAP es una especificación de protocolo para la información basada en XML que las aplicaciones distribuidas pueden usar para intercambiar información y, por lo tanto, interoperar. A menudo está acompañado por un conjunto de estándares de interoperabilidad de middleware SOA e implementaciones compatibles, denominadas (colectivamente) como WS *. SOAP y WS * juntos definen muchos estándares, incluidos los siguientes:

• *Una infraestructura para la composición del servicio*. SOAP puede utilizar el lenguaje de ejecución de procesos de negocios (BPEL) como una forma de permitir a los desarrolladores expresar procesos de negocios que se implementan como servicios WS *.

• *Transacciones*. Existen varios estándares de servicio web para garantizar que las transacciones se gestionen correctamente: WS-AT, WS-BA, WS-CAF y WS-Transaction.

• *Descubrimiento de servicios*. El lenguaje de Descripción, Descubrimiento e Integración Universal (UDDI) permite a las empresas publicar listas de servicios y descubrirse entre sí.

• *Fiabilidad*. SOAP, por sí mismo, no garantiza la entrega confiable de mensajes. Las aplicaciones que requieren tales garantías deben usar servicios que cumplan con el estándar de confiabilidad de SOAP: WS-Reliability.

SOAP es bastante general y tiene sus raíces en un modelo de llamada a procedimiento remoto (RPC) de aplicaciones que interactúan, aunque otros modelos son ciertamente posibles. SOAP tiene un sistema de tipo simple, comparable al que se encuentra en los principales lenguajes de programación. SOAP se basa en HTTP y RPC para la transmisión de mensajes, pero podría, en teoría, implementarse sobre cualquier protocolo de comunicación. SOAP no exige los nombres de los métodos de un servicio, el modelo de direccionamiento o las convenciones de procedimientos. Por lo tanto, elegir SOAP compra poca interoperabilidad real entre aplicaciones, es solo un estándar de intercambio de información. Las aplicaciones que interactúan deben acordar *cómo interpretar* la carga útil, que es donde se obtiene la interoperabilidad semántica.

REST, por otro lado, es un estilo arquitectónico basado en cliente-servidor que está estructurado en torno a un pequeño conjunto de operaciones de creación, lectura, actualización y eliminación (CRUD) (denominadas POST, GET, PUT, BORRAR, respectivamente, en el mundo REST) y un esquema de direccionamiento único (basado en un URI o un identificador uniforme de

recursos). REST impone pocas restricciones en una arquitectura: SOAP ofrece integridad; REST ofrece simplicidad.

REST trata sobre la transferencia de estado y estado y ve la web (y los servicios que los sistemas orientados a servicios pueden enlazar) como una gran red de información a la que se puede acceder mediante un solo esquema de direccionamiento basado en URI. No hay una noción de tipo y, por lo tanto, no hay verificación de tipos en REST; depende de las aplicaciones obtener la semántica de interacción correcta.

Debido a que las interfaces REST son tan simples y generales, cualquier cliente HTTP puede comunicarse con cualquier servidor HTTP, utilizando las operaciones REST (POST, GET, PUT, DELETE) sin ninguna configuración adicional. Eso le proporciona interoperabilidad sintáctica, pero, por supuesto, debe haber un acuerdo a nivel de organización sobre lo que realmente hacen estos programas y qué información intercambian. Es decir, la interoperabilidad semántica no está garantizada entre los servicios solo porque ambos tienen interfaces REST.

REST, además de HTTP, está destinado a ser autodescriptivo y, en el mejor de los casos, es un protocolo sin estado. Considere el siguiente ejemplo, en RESTO, de un servicio de directorio telefónico que le permite a alguien buscar una persona, dado un identificador único para esa persona:

Haga clic aquí para ver la imagen del código

```
http://www.XYZdirectory.com/phonebook/UserInfo/99999
```

La misma búsqueda simple, implementada en SOAP, se especificaría como algo como lo siguiente:

Haga clic aquí para ver la imagen del código

```
<? xml version = "1.0"?>
<soap: Sobre xmlns: soap = http: //www.w3.org/2001/
    12 / soap-envelope
 soap: encodingStyle = "http://www.w3.org/      2001/12 /
    soap-encoding ">
  <soap: Body pb =" http://www.XYZdirectory.com/
    phonebook ">
<pb: GetUserInfo>
       <pb: UserIdentifier> 99999 </ pb: UserIdentifier>
    </ pb: GetUserInfo>
  </ soap: Body>
</ soap: Envelope>
```

Un aspecto de la elección entre SOAP y REST es si desea aceptar la complejidad y las restricciones de SOAP + WSDL (el lenguaje de descripción de servicios

web) para obtener una interoperabilidad más estandarizada o si desea evitar los gastos generales mediante el uso de REST, pero quizás sea un beneficio. De menos estandarización. ¿Cuáles son las otras consideraciones?

Un intercambio de mensajes en REST tiene un poco menos de caracteres que un intercambio de mensajes en SOAP. Entonces, una de las ventajas y desventajas en la elección entre REST y SOAP es el tamaño de los mensajes individuales. Para los sistemas que intercambian una gran cantidad de mensajes, otra compensación es entre el rendimiento (favoreciendo REST) y los mensajes estructurados (favoreciendo SOAP).

La decisión de implementar WS * o REST dependerá de aspectos como la calidad de servicio (QoS) requerida: la implementación de WS * tiene un mayor soporte para la seguridad, la disponibilidad, etc. y el tipo de funcionalidad. Una implementación REST, debido a su simplicidad, es más apropiada para la funcionalidad de solo lectura, típica de los mashups, donde existen requisitos e inquietudes mínimos de QoS.

De acuerdo, si está creando un sistema basado en servicios, ¿cómo lo elige? La verdad es que no tiene que hacer una sola elección, de una vez por todas; Cada tecnología es razonablemente fácil de usar, al menos para aplicaciones simples. Y cada uno tiene sus fortalezas y debilidades. Como todo lo demás en arquitectura, todo se trata de las compensaciones; su decisión probablemente dependerá de la forma en que esas concesiones afectan su sistema en su contexto.

- *RK*

6.2. TÁCTICAS PARA LA INTEROPERABILIDAD

La figura 6.2 muestra el objetivo del conjunto de tácticas de interoperabilidad.

Figura 6.2. Gol de tácticas de interoperabilidad.

Identificamos dos categorías de tácticas de interoperabilidad: localizar y administrar interfaces.

Localizar

Solo hay una táctica en esta categoría: *descubrir servicio* . Se utiliza cuando los sistemas que interactúan deben descubrirse en tiempo de ejecución.

• *Descubrir el servicio* . Localice un servicio mediante la búsqueda de un servicio de directorio conocido. (Por "servicio", simplemente nos referimos a un conjunto de capacidades a las que se puede acceder mediante algún tipo de interfaz). Puede haber múltiples niveles de direccionamiento indirecto en este proceso de ubicación, es decir, una ubicación conocida apunta a otra ubicación que a su vez puede ser Buscó el servicio. El servicio se puede ubicar por tipo de servicio, por nombre, por ubicación o por algún otro atributo.

Gestionar interfaces

La gestión de interfaces consta de dos tácticas: *orquestar* y *adaptar la interfaz* .

• *Organizar* . Orchestrate es una táctica que utiliza un mecanismo de control para coordinar, gestionar y secuenciar la invocación de servicios particulares (que podrían ignorarse entre sí). La orquestación se utiliza cuando los sistemas interoperativos deben interactuar de manera compleja para realizar una tarea compleja; Orquestación "scripts" de la interacción. Los motores de flujo de trabajo son un ejemplo del uso de la táctica orquestada. El patrón de diseño del mediador puede cumplir esta función para una orquestación simple. La orquestación compleja se puede especificar en un lenguaje como BPEL.

• *Interfaz a medida* . La interfaz Tailor es una táctica que agrega o elimina capacidades a una interfaz. Se pueden agregar capacidades como traducción, adición de búferes o suavizado de datos. Las capacidades también pueden ser eliminadas. Un ejemplo de eliminación de capacidades es ocultar funciones particulares de usuarios no confiables. El patrón de decorador es un ejemplo de la táctica de interfaz de sastre.

El bus de servicio empresarial que subyace a muchas arquitecturas orientadas a servicios combina ambas tácticas de interfaz de administración.

La figura 6.3 muestra un resumen de las tácticas para lograr la interoperabilidad.

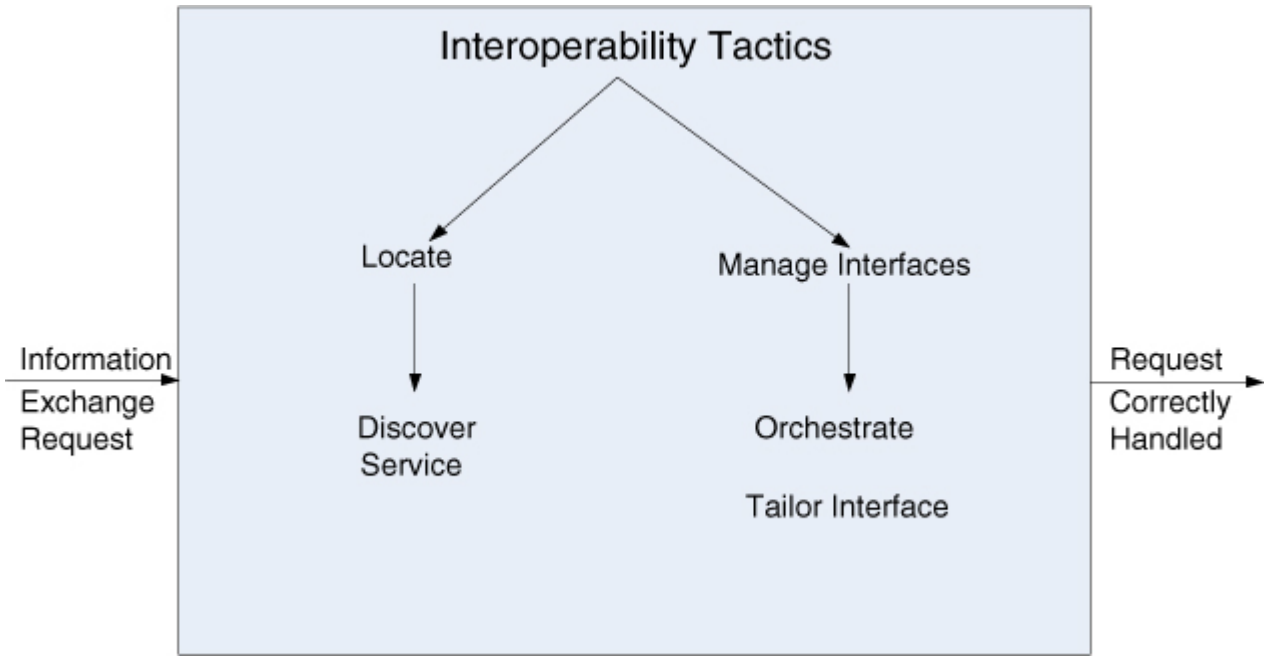

Figura 6.3. Resumen de tácticas de interoperabilidad

Por qué los estándares no son suficientes para garantizar la interoperabilidad *por Grace Lewis*

El Desarrollador del Sistema A necesita intercambiar datos del producto con el Sistema B. El Desarrollador A encuentra que hay una interfaz de servicio web WS * existente para enviar datos del producto que, entre otros campos, contiene el precio expresado en el Esquema XML como un decimal con dos dígitos de fracción. El desarrollador A escribe un código para interactuar con el servicio web y el sistema funciona perfectamente. Sin embargo, después de dos semanas de operación, hay una gran discrepancia entre los totales reportados por el Sistema A y los reportados por el Sistema B. Después de las conversaciones entre los dos desarrolladores, descubren que el Sistema B espera recibir un precio que incluye impuestos y el Sistema A lo estaba enviando sin impuestos.

Este es un ejemplo simple de por qué las normas no son suficientes. Los sistemas intercambiaron datos perfectamente porque ambos acordaron que el precio era un decimal con dos dígitos de fracciones expresados en el esquema XML y el mensaje se envió a través de SOAP a través de HTTP (sintaxis), estándares utilizados en la implementación de servicios web WS *, pero lo hicieron No estoy de acuerdo sobre si el precio incluye impuestos o no (semántica).

Por supuesto, el único enfoque realista para lograr que diversas aplicaciones compartan información es mediante acuerdos sobre la estructura y función de la información que se compartirá. Estos acuerdos a menudo se reflejan en estándares que proporcionan una interfaz común que son compatibles con múltiples proveedores y creadores de aplicaciones. Las normas han sido de

hecho instrumentalespara lograr un nivel significativo de interoperabilidad en el que confiamos en casi todos los dominios. Sin embargo, si bien los estándares son útiles y en muchos aspectos indispensables, las expectativas de lo que se puede lograr a través de los estándares no son realistas. Estos son algunos de los desafíos que enfrentan las organizaciones relacionados con los estándares y la interoperabilidad:

1. Idealmente, cada implementación de un estándar debería ser idéntica y, por lo tanto, completamente interoperable con cualquier otra implementación. Sin embargo, esto está lejos de la realidad. Los estándares, cuando se incorporan a productos, herramientas y servicios, se someten a personalizaciones y extensiones porque cada proveedor desea crear un punto de venta único como una ventaja competitiva.

2. Las normas a menudo son deliberadamente abiertas y proporcionan puntos de extensión. La implementación real de estos puntos de extensión se deja a discreción de los implementadores, lo que lleva a implementaciones propietarias.

3. Las normas, como cualquier tecnología, tienen un ciclo de vida propio y evolucionan con el tiempo de manera compatible y no compatible. Decidir cuándo adoptar una norma nueva o revisada es una decisión crítica para las organizaciones. Comprometerse con un nuevo estándar que no está listo o que finalmente no es adoptado por la comunidad es un gran riesgo para las organizaciones. Por otra parte, esperar demasiado tiempo también puede convertirse en un problema, lo que puede llevar a productos no compatibles, incompatibilidades y soluciones alternativas, porque todos los demás están usando el estándar.

4. Dentro de la comunidad de software, hay tantos estándares malos como ingenieros con opiniones. Los estándares incorrectos incluyen estándares subespecificados, sobre especificados, especificados de manera inconsistente, inestables o irrelevantes.

5. Es bastante común que los estándares sean defendidos por organizaciones competidoras, lo que resulta en estándares conflictivos debido a la superposición o exclusión mutua.

6. Para los dominios nuevos y que emergen rápidamente, el argumento a menudo es que la estandarización será destructiva porque dificultará la flexibilidad: la estandarización prematura forzará el uso de un enfoque inadecuado y conducirá a abandonar otros enfoques probablemente supuestamente mejores. Entonces, ¿qué hacen las organizaciones mientras tanto?

Lo que ilustran estos desafíos es que, debido a la forma en que los estándares se crean y evolucionan, no podemos dejar que los estándares impulsen nuestras arquitecturas. Primero debemos diseñar los sistemas y luego decidir qué estándares pueden soportar los requisitos y cualidades deseados del

sistema. Este enfoque permite que los estándares cambien y evolucionen sin afectar la arquitectura general del sistema.

Una vez escuché a alguien en un discurso de apertura decir que "Lo bueno de los estándares es que hay muchos para elegir".

6.3. UNA LISTA DE VERIFICACIÓN DE DISEÑO PARA LA INTEROPERABILIDAD

La Tabla 6.3 es una lista de verificación para respaldar el proceso de diseño y análisis para la interoperabilidad.

Tabla 6.3. Lista de verificación para apoyar el proceso de diseño y análisis de interoperabilidad

Category	Checklist
Allocation of Responsibilities	Determine which of your system responsibilities will need to interoperate with other systems. Ensure that responsibilities have been allocated to detect a request to interoperate with known or unknown external systems. Ensure that responsibilities have been allocated to carry out the following tasks: ▪ Accept the request ▪ Exchange information ▪ Reject the request ▪ Notify appropriate entities (people or systems) ▪ Log the request (for interoperability in an untrusted environment, logging for nonrepudiation is essential)
Coordination Model	Ensure that the coordination mechanisms can meet the critical quality attribute requirements. Considerations for performance include the following: ▪ Volume of traffic on the network both created by the systems under your control and generated by systems not under your control ▪ Timeliness of the messages being sent by your systems ▪ Currency of the messages being sent by your systems ▪ Jitter of the messages' arrival times ▪ Ensure that all of the systems under your control make assumptions about protocols and underlying networks that are consistent with the systems not under your control.

Data Model	Determine the syntax and semantics of the major data abstractions that may be exchanged among interoperating systems. Ensure that these major data abstractions are consistent with data from the interoperating systems. (If your system's data model is confidential and must not be made public, you may have to apply transformations to and from the data abstractions of systems with which yours interoperates.)
Mapping among Architectural Elements	For interoperability, the critical mapping is that of components to processors. Beyond the necessity of making sure that components that communicate externally are hosted on processors that can reach the network, the primary considerations deal with meeting the security, availability, and performance requirements for the communication. These will be dealt with in their respective chapters.
Resource Management	Ensure that interoperation with another system (accepting a request and/or rejecting a request) can never exhaust critical system resources (e.g., can a flood of such requests cause service to be denied to legitimate users?). Ensure that the resource load imposed by the communication requirements of interoperation is acceptable. Ensure that if interoperation requires that resources be shared among the participating systems, an adequate arbitration policy is in place.
Binding Time	Determine the systems that may interoperate, and when they become known to each other. For each system over which you have control: ▪ Ensure that it has a policy for dealing with binding to both known and unknown external systems. ▪ Ensure that it has mechanisms in place to reject unacceptable bindings and to log such requests. ▪ In the case of late binding, ensure that mechanisms will support the discovery of relevant new services or protocols, or the sending of information using chosen protocols.
Choice of Technology	For any of your chosen technologies, are they "visible" at the interface boundary of a system? If so, what interoperability effects do they have? Do they support, undercut, or have no effect on the interoperability scenarios that apply to your system? Ensure the effects they have are acceptable. Consider technologies that are designed to support interoperability, such as web services. Can they be used to satisfy the interoperability requirements for the systems under your control?

6.4. RESUMEN

La interoperabilidad se refiere a la capacidad de los sistemas para intercambiar información de manera útil. Es posible que estos sistemas se hayan construido con la intención de intercambiar información, pueden ser sistemas existentes que se desean intercambiar información o pueden proporcionar servicios generales sin conocer los detalles de los sistemas que desean utilizar esos servicios.

El escenario general de interoperabilidad proporciona los detalles de estos diferentes casos. En cualquier caso de interoperabilidad, el objetivo es intercambiar información intencionalmente o rechazar la solicitud de intercambio de información.

Para lograr la interoperabilidad, los sistemas relevantes se ubican entre sí y luego administran las interfaces para que puedan intercambiar información.