KAPITOLA 15

Servisně orientovaný návrh (část 3: Návrh služeb)

Ještě před tím, než se můžeme pustit do vývoje služby, musíme definovat její rozhraní. Jedná se o klasické zaříkávadlo všeobecně přijímaného přístupu ve stylu „nejdříve WSDL“ pro návrh webových služeb, a zároveň jde o základ pro každý ze tří procesů návrhu služeb, které si popíšeme v této kapitole. Definování rozhraní služby před vlastním vývojem je důležité pro ustavení vysoce standardizované servisně orientované architektury a vyžaduje realizaci několika charakteristik, které jsme identifikovali jako součásti současné SOA.

Konkrétně lze vytvořením kontraktu služby před její logikou získat následující výhody:

- Služby mohou být navrženy tak, aby přesně reprezentovaly kontext funkci jim odpovídajících kandidátů služeb.
- Na názvy operací služeb lze aplikovat jisté konvence, což vede ke standardizovaným definicím koncových bodů.
- Členění operací může být modelováno abstraktně, čímž získáme konzistentní a předvídatelné návrhy rozhraní, které mimo jiné ustavují poměr velikosti a objemu zpráv vhodný pro cílovou komunikační infrastrukturu.
- Základní aplikace se musejí přizpůsobit stylu návrhu služeb a ne naopak. (To vede často k tvorbě fasádní řídicí vrstvy, která musí komponovat starší komponenty spoléhající na komunikaci v RPC stylu.)
- S návrhem řídicích služeb mohou pomáhat podnikoví analytici, kteří se postarají o přesnou reprezentaci řídicí logiky.

Popisy procesů nabízené v této kapitole jsou ve své povaze obecné a navrhují pouze série určitých kroků pro dokončení návrhu rozhraní služeb. Je třeba na ně pohlížet jako na výchozí body, od nichž mohou organizace odvodit své vlastní, na míru šité návrhové procesy.

Případové studie: Případové studie budou mít v této kapitole vyšší význam, neboť jejich prostřednictvím se seznámíme s ukázkami ve značkovacích jazycích. Jistou podmnožinu kandidátů služeb vytvořených v kapitole 11 v podstatě protáhneme návrhovými procesy stanovenými v této kapitole.

Jako součást těchto procesů vytvoříme nejrůznější WSDL definice a s nimi spojená XSD schémata. Kompletní verze těchto souborů lze stáhnout na adrese *www.serviceoriented.ws*.

Poznámka

Výsledné WSDL dokumenty a XSD schémata z těchto příkladů byly testovány proti verzi 1.1 specifikace WS-I Basic Profile.

Přehled návrhu služeb

Konečným cílem těchto procesů je dosažení vyváženého návrhu služby. To obvykle představuje jistou webovou službu, která vyhovuje požadavkům a omezením reálného světa, a přitom zvládá:

- zapouzdřit požadované množství logiky,
- podřídit se principům servisní orientace,
- vyhovět podnikovým požadavkům.

Nyní se možná díváte na svůj seznam kandidátů služeb a přemýšlíte, kde vlastně začít. Je dobré si tuto otázku položit, neboť existuje preferovaná posloupnost pro tvorbu návrhů služeb. Obecné pravidlo zní: začněte návrhově agnostickými, znovupoužitelnými službami. To umožňuje službám, jež představují logiku specifickou pro nějaký proces, komponovat znovupoužitelné služby jako fixní prostředky (což také prověří kvalitu jejich znovupoužitelnosti).

V souladu se čtyřmi hlavními typy vrstev se službami, které jsme dříve identifikovali, vypadá doporučená posloupnost při návrhu následujícím způsobem:

1. entitně zaměřené řídicí služby (kapitola 15),
2. aplikační služby (kapitola 15),
3. úlohově zaměřené řídicí služby (kapitola 15),
4. procesní služby (kapitola 16).

Tato posloupnost je ve skutečnosti více než jen jakýmsi vodítkem, neboť proces návrhu služeb ve skutečnosti není vždy tak jasný. Například po vytvoření počáteční sady návrhů aplikačních služeb postoupíte k budování úlohově zaměřených služeb. Jenomže až při začleňování nejrůznějších operací zjistíte, že pro jejich provedení potřebujete další funkce na úrovni aplikačních služeb. V důsledku pak musíte revidovat návrhy aplikačních služeb, abyste určili, zdali máte přidat operace nebo zcela nové služby.

Návrhové standardy

Je důležité poznamenat, že pro dosažení úspěšné SOA je stálá sada návrhových standardů naprosto kritická. Vzhledem k tomu, že návrh, který definujeme jako část této fáze, je konkrétní a permanentní, musí být každá vytvořená služba maximálně konzistentní. V opačném případě budou klíčové výhody SOA, jako je znovupoužitelnost, komponovatelnost a především pružnost, ohroženy. Proto se předpokládá, že před zahájením této fáze jsou již návrhové standardy stanoveny. (Část *Pravidla pro návrh služeb* na konci této kapitoly poskytuje určitá doporučení, jež mohou pomoci při formování základů těchto standardů.)

V našem předchozím procesu servisně orientované analýzy jsme na návrhové standardy nekladli tak silný důraz (nutnost standardů jsme sice zmínili, neustavili jsme je však jako součást samotného procesu). To je především dáno tím, že kandidáti služeb mohou být i po vývoji a implementaci odpovídajících služeb bez výrazného dopadu dále modifikováni a vylepšováni. Standardy jsou pro servisně orientovanou analýzu stále důležité, ne však tolik, jako pro servisně orientovaný návrh, jehož jsou integrální součástí.

O popisech procesů

Ukázkové procesy se v této části skládají z obecných sad kroků, jež zdůrazňují primární činitele pro tvorbu návrhů služeb. Jedná se o naši poslední šanci pro zajištění, aby služby náležitě vyjadřovaly svůj účel a schopnosti.

U každého vytvořeného popisu abstraktní služby definujeme formálně následující části:

- definice všech operací služby,
- definice každé vstupní a výstupní zprávy operace,
- definice souvisejících typů XSD schématu používaných pro reprezentaci obsahu zprávy.

Všimněte si, že jednotlivé návrhy služeb složíme později (v kapitole 16) do definice WS-BPEL procesů.

Předpoklady

Jak jsme si vysvětlili v kapitole 13, naše procesy návrhu služeb přistupují k tvorbě rozhraní služeb z perspektivy manuálního programování. To znamená, že během popisů procesů se nevyhneme řadě odkazů na značkovací jazyky WSDL a XSD schéma.

Kromě toho pro podporu našich procesů poskytují četné ukázky případových studií skutečné příklady z oblasti značkovacích jazyků WSDL a XSD schéma. Pro co nejhlubší pochopení popisů procesů a s nimi souvisejících příkladů je tedy vhodné pročíst tutoriály týkající se jazyků WSDL a XSD nabízené kapitolou 13.

PŘÍPADOVÉ STUDIE

Obě naše smyšlené organizace dokončily fázi servisně orientované analýzy pro svou SOA. Během toho obě ustanovily určitou sadu kandidátů služeb, které budou tvořit základ pro jejich servisně orientovaný návrh. Pojďme se nyní podívat, které z těchto kandidátů začleníme do příkladů roztroušených v této kapitole.

Cvičení v modelování služeb provedené společností TLS vedlo k vytvoření následujících pěti nových kandidátů služeb, jež představují nové řešení Předložení pracovních výkazů:

- proces Předložení pracovních výkazů (proces),
- Zaměstnanec (entitně zaměřená služba),
- Pracovní výkaz (entitně zaměřená služba),
- Faktura (entitně zaměřená služba),
- Upozornění (aplikační služba).

Podle navrhované posloupnosti při návrhu bude společnost TLS podrobovat odpovídajícímu procesu návrhu služeb nejdříve tři výše uvedené entitně zaměřené služby. Poté navrhne službu Upozornění společně s dalšími požadovanými aplikačními službami. Nakonec při návrhu procesní služby Předložení pracovních výkazů pomocí definice WS-BPEL procesu definuje vlastní logiku obchodního procesu.

Během všech těchto činností se naše příklady zaměří na návrh následujících dvou služeb:

- Na kandidáta služby Zaměstnanec se budeme odkazovat v příkladech, které jsou součástí popisu procesu *Návrh entitně zaměřených řídicích služeb*.
- Kandidát procesní služby Předložení pracovních výkazů se stane základem pro logiku pracovního toku nezbytného k ustavení náležité kompozice služeb pro proces Předložení pracovních výkazů. (Společnost TLS toho docílí v kapitole 16.)

Mezi kandidáty služeb modelované společností RailCo patří následující čtyři aplikační služby a dvě úlohově zaměřené řídicí služby.

- Zpracování faktur (úlohově zaměřená služba).
- Zpracování objednávek (úlohově zaměřená služba).
- Původní systém (aplikační služba).
- Upozornění na aktualizace (aplikační služba).
- Převod účetních dokumentů (aplikační služba).
- Kontrola metadat (aplikační služba).

V příkladech našeho popisu procesu návrhu budeme pracovat s dvěma kandidáty služeb:

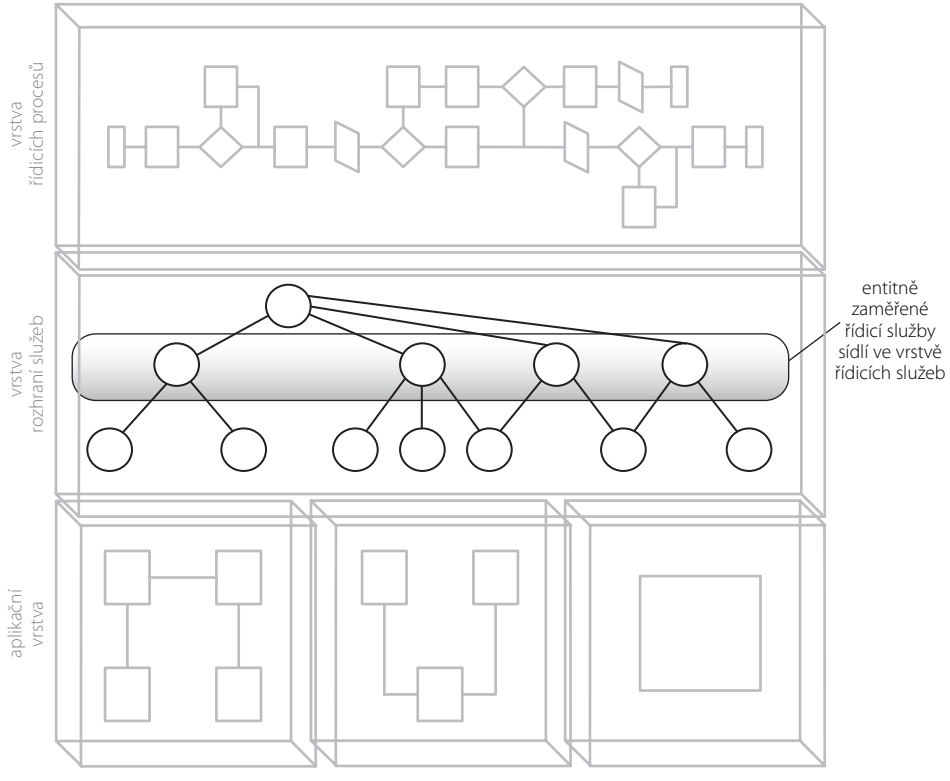
- Pomocí kandidáta služby Převod účetních dokumentů demonstrujeme popis procesu *Návrh aplikačních služeb*.
- Kandidáta služby Zpracování faktur použijeme v části *Návrh úlohově zaměřených řídicích služeb*.

SHRNUTÍ HLAVNÍCH POZNATKŮ

- Tři návrhové procesy popsané v této kapitole jsou založené na přístupu „nejdříve WSDL".
- Návrhové standardy hrají kritickou roli při utváření procesu návrhu služeb a při zajištění konzistentně standardní SOA.

Návrh entitně zaměřených řídicích služeb (krok za krokem)

Entitně zaměřené řídicí služby představují právě tu vrstvu služeb, která je nejméně ovlivňována ostatními. Jejím účelem je přesně reprezentovat odpovídající datové entity definované v rámci obchodních modelů organizace. Tyto služby jsou vzhledem k řešení a obchodnímu procesu striktně agnostické, budují se pro opětovné použití jakoukoli aplikací, jež potřebuje přistupovat nebo spravovat informace spojené s konkrétní entitou.

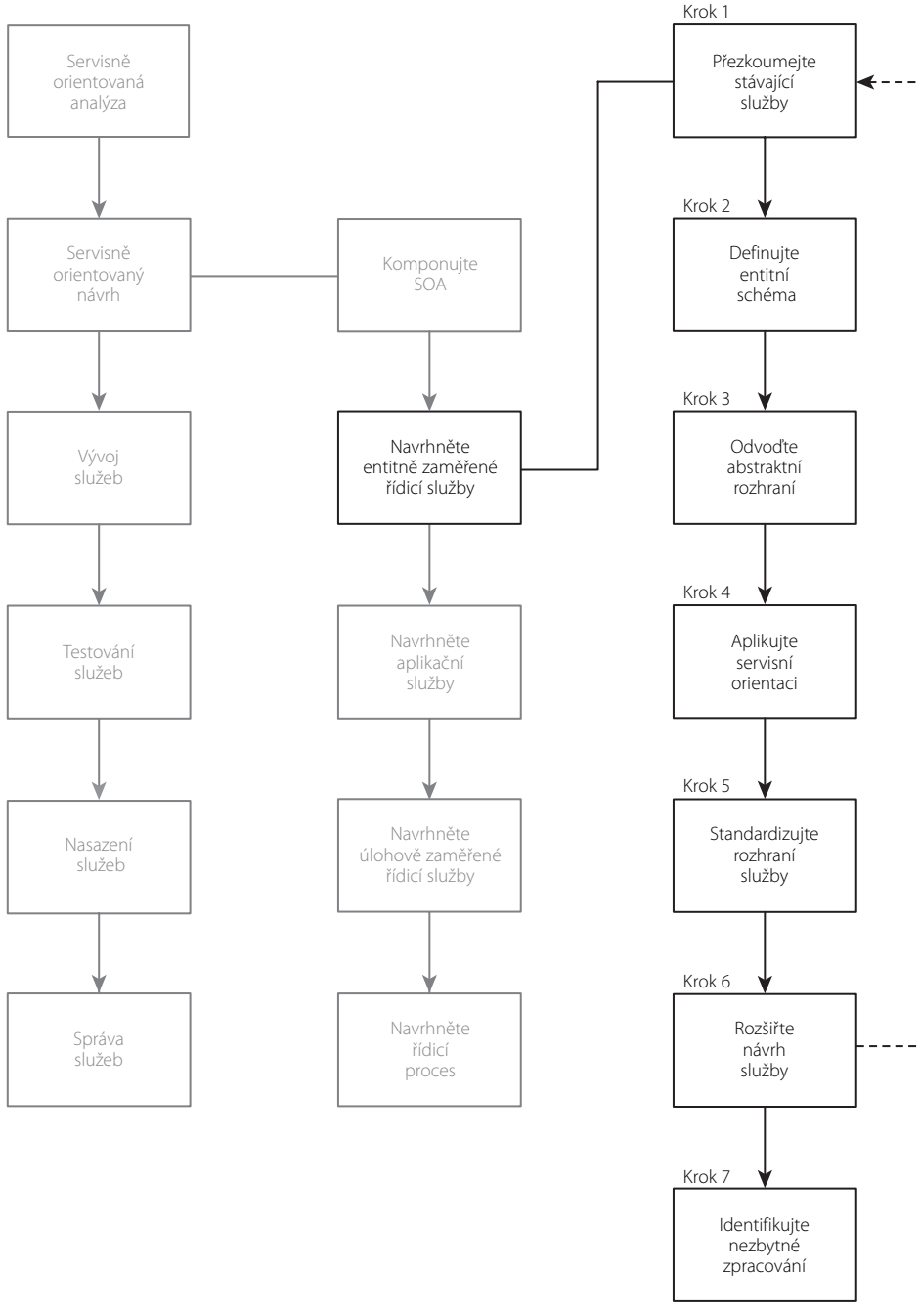


Obrázek 15.1. Entitně zaměřené služby utvářejí vrstvu řídicích služeb

Díky tomu, že ve vztahu k ostatním vrstvám služeb existují spíše atomicky, je výhodné navrhovat entitně zaměřené řídicí služby před ostatními. Tím se ustaví abstraktní vrstva služeb, kolem níž lze rozmístit procesní i základní aplikační logiku.

Popis procesu

Následuje popis procesu vedený krok za krokem, v rámci něhož ustavíme doporučovanou posloupnost detailních kroků pro dosažení kvalitního rozhraní entitně zaměřených řídicích služeb (viz obrázek 15.2).



Obrázek 15.2. Návrhový proces entitně zaměřených řídicích služeb

Všimněte si, že pořadí, v němž se tyto kroky objevují, není pevně dáno. Můžete tak například definovat předběžné rozhraní služby před stanovením skutečných datových typů používaných pro reprezentaci obsahu těla zprávy. Nebo můžete shledat efektivnějším provést spekulativní analýzu k identifikaci možných rozšíření pro služby před prvním nástinem jejího rozhraní.

Všechny výše uvedené přístupy jsou legitimní. Klíčem je zajistit, aby se návrhové standardy nakonec aplikovaly rovnoměrně na všechny operace služeb, a aby se přesně identifikovaly veškeré požadavky na zpracování.

Začněme tedy návrhem našich entitně zaměřených řídicích služeb.

PŘÍPADOVÉ STUDIE

V příkladech nabízených vedle popisu tohoto procesu opět navštívíme prostředí společnosti TLS. Konkrétně se znovu podíváme na kandidáta služby Zaměstnanec, kterého jsme vymodelovali na konci kapitoly 12 (viz obrázek 15.3).

Obrázek 15.3. Kandidát služby Zaměstnanec

Služba Zaměstnanec byla vymodelována záměrně, aby usnadnila seskupování entitně zaměřených operací. Jak součást procesu Předložení pracovních výkazů je nutné, aby přispívala dvěma specifickými funkcemi.

První vyžaduje, aby se provedl dotaz na záznam zaměstnance pro získání maximálního počtu hodin, které smí daný zaměstnanec odpracovat za týden. Druhá část funkcionality poskytuje možnost posílat aktualizace do historie zaměstnance. Jak si možná vybavíte z původního procesu Předložení pracovních výkazů, tato akce je nezbytná pouze tehdy, když je pracovní výkaz zamítnut.

Výsledkem procesu modelování služeb společnosti TLS bylo vyjádření těchto dvou funkcí prostřednictvím přiřazení následujících kandidátů operací:

- vezmi týdenní hodinový limit,
- aktualizuj historii zaměstnance.

Kandidát služby nám nyní poskytuje primární vstup, z něhož odvodíme návrh služby tak, že budeme postupovat podle následujících kroků návrhového procesu entitně zaměřené řídicí služby.

Poznámka

Kandidát operace „vezmi týdenní hodinový limit" (z něhož se později stane operace getWeeklyHoursLimit) představuje jistým způsobem neobvykle jemně členěnou operaci. Operace služeb mají obecně sklon k hrubšímu členění, aby překonaly vyšší režii spojenou s výměnou zpráv protokolu SOAP. Pro zachování jednoduchosti nebudeme členění této operace měnit, neboť naplňuje funkční požadavky našeho obchodního procesu. Pro více informací souvisejících s členěním rozhraní služby nahlédněte do pravidla *Aplikujte vhodnou úroveň členění rozhraní* na konci této kapitoly.

Krok 1: Přezkoumejte stávající služby

Při tvorbě entitně zaměřených služeb bere modelování vedoucí ke kandidátům služeb v ideálním případě v potaz jakékoli existující služby. Ovšem vzhledem k tomu, že kandidáti služeb mají sklon obsahovat kandidáty operací související s podnikovými požadavky, jež formují základ servisně orientované analýzy, stojí vždy za to zajistit, aby některé nebo všechny procesní funkční prvky reprezentované kandidáty operací nebyly také součástí jiných služeb.

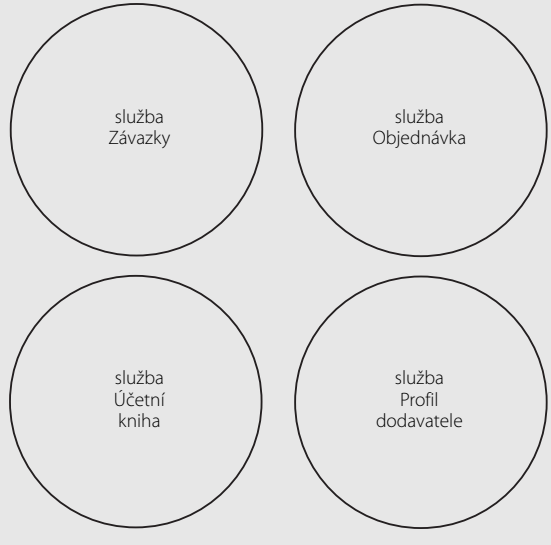
Z tohoto důvodu je prvním krokem při návrhu nové služby ověření, je-li skutečně nezbytná. Pokud existují i jiné služby, pak již mohou poskytovat některé nebo všechny funkční prvky identifikované v kandidátech operací, nebo již mohly ustavit vhodný kontext, v rámci něhož lze tyto nové kandidáty operací implementovat (jako nové operace stávajících služeb).

PŘÍPADOVÉ STUDIE

Jedná se teprve o druhé servisně orientované řešení společnosti TLS. První bylo vytvořeno pro ustavení externího rozhraní s jejich účetním systémem přes prostředí B2B. Toto řešení bylo vybudováno podle vývojové strategie shora dolů, a proto je výsledkem kolekce entitně zaměřených řídicích služeb.

Architekti zapojení do návrhu služeb nové aplikace Předložení pracovních výkazů jsou si docela jisti, že nová sada služeb se s těmi stávajícími žádným způsobem nekryje. Nicméně, vzhledem k tomu, že řešení B2B bylo sestaveno úplně jiným projektovým týmem, architekti souhlasí, že před zahájením návrhového procesu stojí za námahu přezkoumat existující služby.

Jejich průzkum odhalil, že následující entitně zaměřené řídicí služby byly dodány jako součást projektu B2B (viz obrázek 15.4): služba Závazky (Accounts Payable Service), služba Objednávky (Purchase Order Service), služba Účetní kniha (Ledger Service) a služba Profil dodavatele (Vendor Profile Service).



Obrázek 15.4. Stávající inventář společnosti TLS

Již podle samotného pojmenování je patrné, že každá služba představuje určitou entitu odlišnou od entity Zaměstnanec navrhované současným kandidátem služby. Pro jistotu ještě prozkoumají popisy každé služby (spolu s doplňujícími metadaty). Projektoví architekti poté došli k závěru, že k žádnému překrývání nedochází, což jim umožnilo pokračovat v další práci na službě Zaměstnanec.

Krok 2: Definujte entitní schéma

Je užitečné začít návrh rozhraní určité služby formální definicí zpráv, které má služba zpracovávat. K tomuto účelu musíme formalizovat struktury zpráv, jež jsou definovány v rámci WSDL konstruktů `types`.

SOAP zprávy přenášejí datový obsah v rámci sekce `Body` SOAP obálky. Tato data je nutné uspořádat a otypovat. Pro tento účel se spoléháme na XSD schémata. Samostatné schéma lze vložit do konstruktu `types`, v rámci něhož můžeme definovat každý element představující data v těle SOAP zprávy.

V případě entitně zaměřených služeb je zvláště užitečné, pokud použité XSD schéma přesně reprezentuje informace spojené s entitou služby. Toto „entitně zaměřené schéma" se může stát základem pro WSDL definici služby, neboť u většiny operací služeb se očekává, že přijímají či vysílají dokumenty definované tímto schématem.

Všimněte si, že mezi entitně zaměřenými službami a entitami tvořícími entitní model nemusí nutně být vzájemně rovnocenný vztah. Možná si vybavíte, že jsme v příkladu modelování služby v kapitole 12 kombinovali entity Zaměstnanec a Historie zaměstnance do jediné služby Zaměstnanec. V takovém případě můžete buď vytvořit dvě samostatná schémata, nebo je zkombinovat do jediného. Druhá možnost je doporučovaná pouze tehdy, když jste si jisti, že již nikdy nebudete chtít tyto entity znovu rozdělit.

Poznámka

Jak si ukážeme v nadcházejícím příkladu, WSDL definice umí importovat schémata do oblasti `types`. To může být užitečné zvláště tehdy, když pracujeme se standardizovanými schématy, jež představují entity. (Více informací najdete v pravidle *Zvažte používání modulárních WSDL dokumentů*.)

PŘÍPADOVÉ STUDIE

Společnost TLS již před nějakou dobou investovala do vytvoření standardizované architektury pro reprezentaci XML dat (pouze pro jejich účetní prostředí). To znamená, že inventář entitně zaměřených XSD schémat představujících informace související s účetnictvím již existuje.

Na první pohled to vypadá, že tím pádem bude tento krok snazší. Nicméně po bližším prostudování bylo objeveno, že stávající XSD schéma je velmi rozsáhlé a složité. Po určité diskusi se architekti společnosti TLS rozhodli, že existující schéma s touto službou nepoužijí. Místo toho se rozhodli dodat odlehčenou (ale přitom plně vyhovující) verzi tohoto schématu, aby tak lépe vyšli vstříc jednoduchým procesním požadavkům služby Zaměstnanec.

Začali identifikací druhů dat, jež bude třeba vyměňovat pro naplnění procesních požadavků kandidáta operace „vezmi týdenní hodinový limit“. Skončili definováním dvou složených typů: jeden obsahuje vyhledávací kritéria nezbytná pro správu požadavku přijatého službou Zaměstnanec a druhý obsahuje výsledky dotazu vrácené touto službou. Tyto typy jsou náležitě pojmenovány, takže je patrná jejich spojitost s příslušnými zprávami. Tyto dva typy pak tvoří nový soubor schématu Employee.xsd.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace=
      "http://www.xmltc.com/tls/employee/schema/accounting/">
   <xsd:element name="EmployeeHoursRequestType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ID" type="xsd:integer"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="EmployeeHoursResponseType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ID" type="xsd:integer"/>
            <xsd:element name="WeeklyHoursLimit" type="xsd:short"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Výpis 15.1. Schéma Employee (zaměstnanec) poskytující konstrukty `complexType` použité pro stanovení předpokládané datové reprezentace pro kandidáta operace „vezmi týdenní hodinový limit“

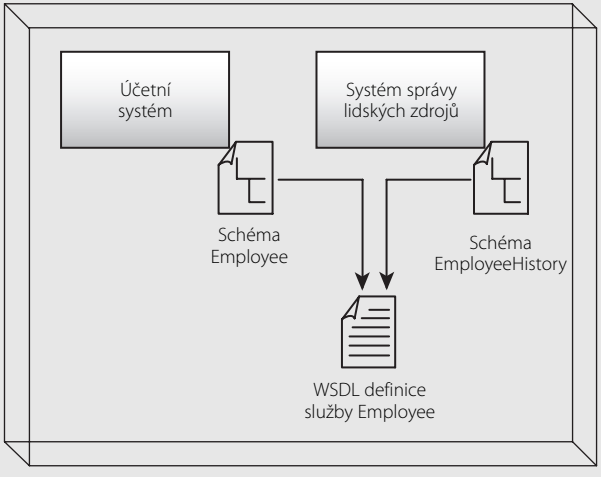
Poznámka

Konstrukty `complexType` jsou obalené konstrukty `element`, aby vyhovovaly požadavkům specifikace WS-I pro SOAP zprávy ve stylu dokument + literál.

Jakmile se však architekti pokoušeli odvodit typy nezbytné pro kandidáta operace „aktualizuj historii zaměstnance“, objevil se jiný problém.

Objevili totiž, že schéma, od něhož odvodili soubor Employee.xsd, nereprezentuje entitu Historie zaměstnance, kterou tento kandidát služby také zapouzdřuje.

Další návštěva archivu s účetním schématem odhalila, že informace o historii zaměstnance není řízena účetním řešením, ale je součástí prostředí pro správu lidských zdrojů, pro něž nebylo vytvořeno žádné schéma.



Obrázek 15.5. Dvě schémata pocházející ze dvou různých datových zdrojů

Aby nebyl narušen již standardizovaný návrh schématu Employee (zaměstnanec), tak bylo rozhodnuto, že se vytvoří druhá definice schématu (viz obrázek 15.5). Zapojili se také analytici a vytvořili následující schéma EmployeeHistory.xsd:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace=
      "http://www.xmltc.com/tls/employee/schema/hr/">
   <xsd:element name="EmployeeUpdateHistoryRequestType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ID" type="xsd:integer"/>
            <xsd:element name="Comment" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="EmployeeUpdateHistoryResponseType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ResponseCode" type="xsd:byte"/>
```

```
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Výpis 15.2. Schéma EmployeeHistory (historie zaměstnance) s odlišnou hodnotou atributu `targetNamespace` pro identifikace jeho rozdílného původu

K podpoře znovupoužitelnosti a pro umožnění, aby byl každý soubor schématu udržován odděleně od WSDL dokumentu, se pomocí příkazů XSD schématu `import` dodají obsahy obou schémat do WSDL konstruktu `types` služby Zaměstnanec.

```
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/tls/employee/schema/">
      <xsd:import namespace=
         "http://www.xmltc.com/tls/employee/schema/accounting/"
         schemaLocation="Employee.xsd"/>
      <xsd:import namespace=
         "http://www.xmltc.com/tls/employee/schema/hr/"
         schemaLocation="EmployeeHistory.xsd"/>
   </xsd:schema>
</types>
```

Výpis 15.3. WSDL konstrukt `types` naplněný importovanými schématy

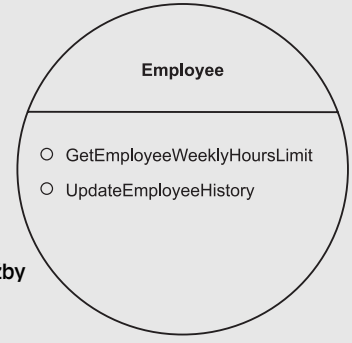
Krok 3: Odvoďte abstraktní rozhraní

V dalším kroku analyzujeme navrhované kandidáty operací služeb a podle následujících kroků definujeme rozhraní služby:

1. Ověřte, že každý kandidát operace je vhodným způsobem generický a znovupoužitelný, a to tak, že zajistíte patřičné členění zapouzdřené logiky. Prostudujte datové struktury definované v kroku 2 a stanovte sadu názvů operací.
2. Uvnitř WSDL-dokumentu vytvořte oblast `portType` (či `interface`) a naplňte ji konstrukty `operation`, jež odpovídají kandidátům operací.
3. Formalizujte seznam vstupních a výstupních hodnot nezbytných pro správné zpracování logiky každé operace. Toho docílíte tak, že definujete náležité konstrukty `message`, jež odkazují na typy XSD-schématu v rámci podřízených elementů `part`.

PŘÍPADOVÉ STUDIE

Architekti společnosti TLS se dohodli na následujících názvech operací: `GetEmployeeWeeklyHoursLimit` **a** `UpdateEmployeeHistory` **(viz obrázek 15.6).**



Obrázek 15.6. Operace služby Employee (zaměstnanec)

Poté postoupili k definici zbývajících částí abstraktní definice, tedy konkrétně ke konstruktům `message` **a** `portType`.

```
<message name="getEmployeeWeeklyHoursRequestMessage">
   <part name="RequestParameter"
      element="act:EmployeeHoursRequestType"/>
</message>
<message name="getEmployeeWeeklyHoursResponseMessage">
   <part name="ResponseParameter"
      element="act:EmployeeHoursResponseType"/>
</message>
<message name="updateEmployeeHistoryRequestMessage">
   <part name="RequestParameter"
      element="hr:EmployeeUpdateHistoryRequestType"/>
</message>
<message name="updateEmployeeHistoryResponseMessage">
   <part name="ResponseParameter"
      element="hr:EmployeeUpdateHistoryResponseType"/>
</message>
<portType name="EmployeeInterface">
   <operation name="GetEmployeeWeeklyHoursLimit">
      <input message="tns:getEmployeeWeeklyHoursRequestMessage"/>
      <output message="tns:getEmployeeWeeklyHoursResponseMessage"/>
```

```
    </operation>
    <operation name="UpdateEmployeeHistory">
       <input message="tns:updateEmployeeHistoryRequestMessage"/>
       <output message="tns:updateEmployeeHistoryResponseMessage"/>
    </operation>
</portType>
```

Výpis 15.4. Části `message` a `portType` definice služby Employee (zaměstnanec), jež implementují abstraktní podrobnosti definice dvou operací této služby

Poznámka

Společnost TLS postavila standardizaci na specifikaci WSDL 1.1, protože vyhovuje požadavkům diktovaným verzí 1.1 profilu WS-I Basic Profile, a protože žádná z jejích aplikačních platforem nepodporuje novější verzi jazyka WSDL. WSDL 1.1 používá místo elementu `interface` zavedeného od verze WSDL 2.0 element `portType`.

Krok 4: Aplikujte servisní orientaci

V tomto kroku revidujeme čtyři principy servisní orientace, jež jsme identifikovali v kapitole 8 jako ty, které neposkytuje technologická sada webových služeb:

- znovupoužitelnost služeb,
- autonomie služeb,
- bezstavovost služeb,
- zjistitelnost služeb.

Znovupoužitelnost a autonomie, což jsou dva principy, kterým jsme se již věnovali v procesu modelování služeb, jsou určitým způsobem přirozenou součástí entitně zaměřeného návrhového modelu, protože operace předkládané entitně zaměřenými řídicími službami bývají neodmyslitelně generické a znovupoužitelné (a protože se používání příkazu import doporučuje pro opětovné používání schémat a vytváření modulárních WSDL definic). Znovupoužitelnost je dále propagována v kroku 6, kde navrhujeme rozšíření návrhu pro podporu požadavků ležících za hranicí těch, které byly identifikovány jako součást našeho kandidáta služby.

Protože entitně zaměřené služby často musejí být složeny nadřazenou vrstvou služeb, a protože se pro provádění své řídicí logiky opírají o vrstvu aplikačních služeb, je jejich bezprostřední autonomie obecně dobře definována. Dokud nemají tyto služby ovládané entitně zaměřeným řadičem neobvyklé požadavky na zpracování, nebo si nějakým způsobem nevynucují další závislosti, tak si svou autonomii zpravidla udržují.

Bezstavovost je také relativně ovladatelná, a to z podobných, výše uvedených důvodů. Entitně zaměřené služby obecně nevlastní nijak velkou část logiky pracovního toku a pro ty případy, kdy je třeba vyvolat vícero aplikačních či řídicích služeb pro provedení nějaké operace, se preferuje, aby byla správa stavu v co nejvyšší míře odložena (například do SOAP zprávy dokumentového stylu).

Zjistitelnost je důležitou částí jak návrhu entitně zaměřených služeb, tak i jejich užívání po samotném nasazení. Jak jsme zmínili v kroku 1, potřebujeme zajistit, aby návrh služby neimplementoval logiku, která již existuje. Mechanismus zjistitelnosti by toto ověření značně usnadnil. Jednou z věcí, kterou můžeme udělat, aby byla daná služba pro ostatní zjistitelnější, je doplnit ji o podrobná metadata pomocí elementu `documentation`, jak si vysvětlíme v pravidle *Dokumentujte služby pomocí metadat*.

PŘÍPADOVÉ STUDIE

Po přezkoumání počátečních abstraktních rozhraní služeb bylo rozhodnuto, že pro lepší podporu základních aspektů servisní orientace lze provést menší revizi. Konkrétně se k WSDL definicím přidají metainformace pro lepší popis účelu a funkce každé ze dvou operací a s nimi spojených zpráv.

```
<portType name="EmployeeInterface">
   <documentation>
      Operace GetEmployeeWeeklyHoursLimit (vezmi týdenní hodinový
      limit zaměstnance) používá ID zaměstnance
      pro načtení hodnoty WeeklyHoursLimit (týdenní hodinový limit).
      Operace UpdateEmployeeHistory (aktualizuj historii zaměstnance)
      používá ID zaměstnance pro aktualizaci hodnoty
      Comment (komentář) v rámci entity
      EmployeeHistory (historie zaměstnance).
   </documentation>
   <operation name="GetEmployeeWeeklyHoursLimit">
      <input message="tns:getEmployeeWeeklyHoursRequestMessage"/>
      <output message="tns:getEmployeeWeeklyHoursResponseMessage"/>
   </operation>
   <operation name="UpdateEmployeeHistory">
      <input message="tns:updateEmployeeHistoryRequestMessage"/>
      <output message="tns:updateEmployeeHistoryResponseMessage"/>
   </operation>
</portType>
```

Výpis 15.5. Rozhraní služby doplněné o dodatečnou dokumentaci ve formě metadat

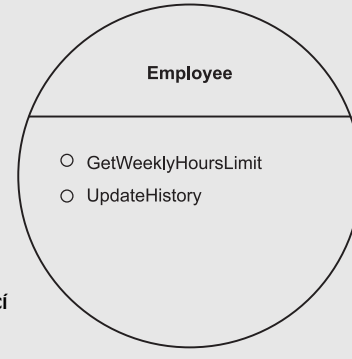
Krok 5: Standardizujte a zdokonalte rozhraní služby

V závislosti na vašich požadavcích může jít o víceaspektový krok zahrnující série návrhových činností. Následuje seznam doporučených akcí, které můžete provést pro dosažení standardizovaného a moderního návrhu:

- Přezkoumejte stávající návrhové standardy a pravidla a všechny vhodné aplikujte. (Jako výchozí bod použijte pravidla a navrhované standardy nabízené na konci této kapitoly.)
- Kromě dosažení standardizovaného návrhu rozhraní služby poskytuje tento krok také příležitost k revizi návrhu služby pro podporu některých charakteristik současné SOA, které jsme identifikovali v části *Nepodporované vlastnosti SOA* v kapitole 9.
- Pokud mezi vaše návrhové požadavky patří splnění specifikace WS-I Basic Profile, pak ji v této fázi může být nezbytné vzít v potaz. I když dodržení profilu Basic Profile vyžaduje, aby byly všechny WSDL definice kompletní, lze verifikovat pouze to, co bylo vytvořeno až do této chvíle.

PŘÍPADOVÉ STUDIE

Architekti společnosti TLS, kteří mají na starosti návrh služby Employee (zaměstnanec), se rozhodli provést úpravy abstraktního rozhraní této služby pro aplikaci současných návrhových standardů. Konkrétně pro standardizaci jmen operací začlenili konvence pro pojmenování, jak znázorňuje obrázek 15.7.



Obrázek 15.7. Revidované názvy operací služby Employee (zaměstnanec)

```
<operation name="GetWeeklyHoursLimit">
   <input message="tns:getWeeklyHoursRequestMessage"/>
   <output message="tns:getWeeklyHoursResponseMessage"/>
</operation>
<operation name="UpdateHistory">
   <input message="tns:updateHistoryRequestMessage"/>
   <output message="tns:updateHistoryResponseMessage"/>
</operation>
```

Výpis 15.6. Dva konstrukty operation s novými, standardizovanými názvy

Jak si vysvětlíme v pravidle *Aplikujte standardy pro pojmenování*, použití standardních názvů poskytuje nativní podporu pro skutečnou interoperabilitu, což je klíčovou charakteristikou současné SOA.

Krok 6: Rozšiřte návrh služby

Proces modelování služeb má sklon zaměřovat se na očividné podnikové požadavky. Zatímco propagace znovupoužitelnosti je vždy podporována, často sklouzává do procesu návrhu, aby bylo zajištěno, že dostatečná míra znovupoužitelnosti bude zapracována do každé služby. To

je zvláště důležité pro entitně zaměřené řídicí služby, neboť jejich žadatelé od nich očekávají kompletní rozsah běžně používaných operací.

Tento krok zahrnuje provedení spekulativní analýzy zaměřené na to, jaké další typy funkcí by daná služba měla v rámci jejího předdefinovaného funkčního kontextu nabízet.

Novou funkcionalitu lze implementovat dvěma obvyklými způsoby:

- přidat nové operace,
- nebo přidat ke stávajícím operacím nové parametry.

Zatímco druhá možnost může zmodernizovat rozhraní služby, může být také neintuitivní, a to v tom, že příliš mnoho parametrů spojených s jedinou operací může vyžadovat, aby žadatelé této operace museli k jejímu efektivnímu využití znát tuto službu přehnaně dobře.

Přidávání operací je přímočarým způsobem, jak poskytnout zřejmé funkce spojené s danou entitou. Do klasické sady operací pro nějakou entitně zaměřenou službu patří:

- GetSomething (něco vezmi),
- UpdateSomething (něco aktualizuj),
- AddSomething (něco přidej),
- DeleteSomething (něco smaž).

Nehledě na bezpečnostní požadavky vytváří zavedení těchto standardních operací konzistentní úroveň interoperability do vrstvy řídicích služeb, což usnadňuje nahodilou znovupoužitelnost a kompozici.

Poznámka

Navzdory výše uvedeným návrhům pro pojmenování je často výhodné při návrhu řídicích služeb odrážejících stávající entitní modely přenést již ustavené konvence pro pojmenování (i když to znamená příslušným způsobem upravit existující standardy pro pojmenování).

Jsou-li definovány úplně nové úlohy, pak mohou být začleněny novými operacemi, jež se drží téhož návrhového standardu jako ty stávající. Jsou-li identifikovány nové funkční požadavky, jež souvisejí s existujícími operacemi, pak je běžnou metodou pro rozšíření těchto operací přidat vstupní a výstupní hodnoty. To umožňuje dané operace přijímat a odesílat pestrou škálu kombinací zpráv. Je však třeba jisté opatrnosti, aby nedošlo k přílišnému zkomplikování operací kvůli potenciální znovupoužitelnosti. Často je vhodné podrobit jakoukoli nově navrhovanou funkcionalitu samostatnému procesu analýzy.

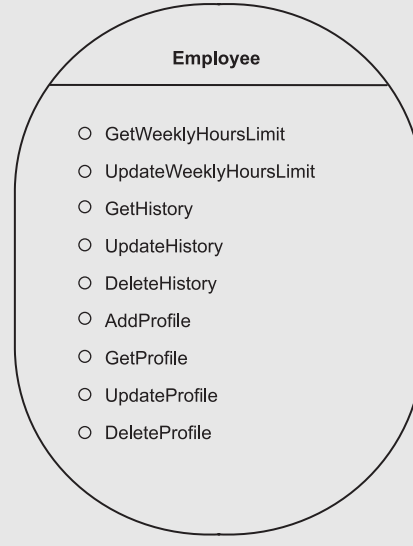
Kromě toho, i když je žádoucí a doporučované vytvářet entitně zaměřené služby, zcela soběstačné při správě dat spojených s odpovídající entitní doménou, existuje klíčový praktický činitel, který je třeba vzít v potaz. Pro každou novou operaci, kterou přidáte, je třeba navrhnout a implementovat také způsob, jakým tato operace dokončí své zpracování. To se redukuje na velmi pravděpodobný požadavek na dodatečné či rozšířené aplikační služby. Dokud jsou režijní požadavky kladené na každou novou operaci odhadnuté a považované za přijatelné, pak je tento krok rozumný.

Všimněte si, že po identifikaci nových operací je pro řádné utvoření a standardizaci přidaných rozšíření nutné opakovat kroky 1 až 5.

PŘÍPADOVÉ STUDIE

Společnost TLS je s dodávkou řešení Předložení pracovních výkazů v časové tísni. Proto je rozhodnuto, že v současné chvíli nebude návrh služby rozšířen. Dosud aplikované standardy jim zaručují snadno rozšiřitelný návrh služby, do něhož mohou přidávat dodatečné operace, aniž by narušili původní rozhraní služby.

Nicméně, pojďme pro demonstraci tohoto kroku uvážit, jaké druhy operací *by mohly* být do služby Employee (zaměstnanec) přidány. Vyjdeme-li z toho, že tato služba představuje dvě entity, pravděpodobně bude vyžadovat mnohem více operací než většina entitně zaměřených služeb, jak znázorňuje obrázek 15.8.



Obrázek 15.8. Služba Employee (zaměstnanec) nabízející ucelený rozsah operací

Následuje příklad, jenž ukazuje, jak by mohl být konstrukt `portType` rozšířen doplňkovými operacemi (pro ušetření místa byly elementy `documentation` vynechány):

```
<portType name="EmployeeInterface">
   <operation name="GetWeeklyHoursLimit">
      <input message="tns:getWeeklyHoursRequestMessage"/>
      <output message="tns:getWeeklyHoursResponseMessage"/>
   </operation>
   <operation name="UpdateWeeklyHoursLimit">
      <input message="tns:updateWeeklyHoursRequestMessage"/>
      <output message="tns:updateWeeklyHoursResponseMessage"/>
   </operation>
   <operation name="GetHistory">
      <input message="tns:getHistoryRequestMessage"/>
      <output message="tns:getHistoryResponseMessage"/>
   </operation>
```

```
    <operation name="UpdateHistory">
       <input message="tns:updateHistoryRequestMessage"/>
       <output message="tns:updateHistoryResponseMessage"/>
    </operation>
    <operation name="DeleteHistory">
       <input message="tns:deleteHistoryRequestMessage"/>
       <output message="tns:deleteHistoryResponseMessage"/>
    </operation>
    <operation name="AddProfile">
       <input message="tns:addProfileRequestMessage"/>
       <output message="tns:addProfileResponseMessage"/>
    </operation>
    <operation name="GetProfile">
       <input message="tns:getProfileRequestMessage"/>
       <output message="tns:getProfileResponseMessage"/>
    </operation>
    <operation name="UpdateProfile">
       <input message="tns:updateProfileRequestMessage"/>
       <output message="tns:updateProfileResponseMessage"/>
    </operation>
    <operation name="DeleteProfile">
       <input message="tns:deleteProfileRequestMessage"/>
       <output message="tns:deleteProfileResponseMessage"/>
    </operation>
</portType>
```

Výpis 15.7. Rozšířený konstrukt portType

Tyto dodatečné operace poskytují pěkně vyzrálou sadu rozšíření pro zpracování dat, jež umožňuje znovupoužití služby Employee (zaměstnanec) v nejrůznějších řešeních.

Krok 7: Identifikujte nezbytné zpracování

I když proces modelování služeb z naší servisně orientované analýzy snad identifikoval některé klíčové aplikační služby, nebylo možné, aby je definoval všechny.

Nyní, když už máme skutečný návrh pro tuto novou řídicí službu, můžete blíže prostudovat procesní požadavky každé z jejích operací. Přitom byste měli být schopni určit, jsou-li pro provedení každé části exponované funkcionality potřebné ještě další aplikační služby. Pokud ano, pak musíte stanovit, zda již existují, nebo zda je třeba je přidat na seznam služeb, které budou dodány jako součást tohoto řešení.

PŘÍPADOVÉ STUDIE

Pojďme se znovu podívat na dvě operace, které jsme navrhli ve službě Employee (zaměstnanec):

- GetWeeklyHoursLimit (vezmi týdenní hodinový limit),
- UpdateHistory (aktualizuj historii).

První vyžaduje přístup k profilu zaměstnance. Ve společnosti TLS jsou informace o zaměstnancích uloženy na dvou místech:

- Mzdová data jsou uchovávána v rámci repozitáře účetního systému společně s kontaktními údaji zaměstnance.
- Údaje o profilu zaměstnance včetně podrobností o jeho historii jsou uloženy v repozitáři pro správu lidských zdrojů.

Když byla ve společnosti TLS poprvé implementována architektura reprezentace XML dat, přemostily se některé ze stávajících nestejností mezi řadou datových zdrojů společnosti TLS pomocí entitně zaměřených XSD schémat. Architekti služeb si toho byli vědomi a pro stanovení procesních požadavků pro operace GetWeeklyHoursLimit (vezmi týdenní hodinový limit) prozkoumali původní schéma Employee.xsd používané jako součást definice Employee.wsdl.

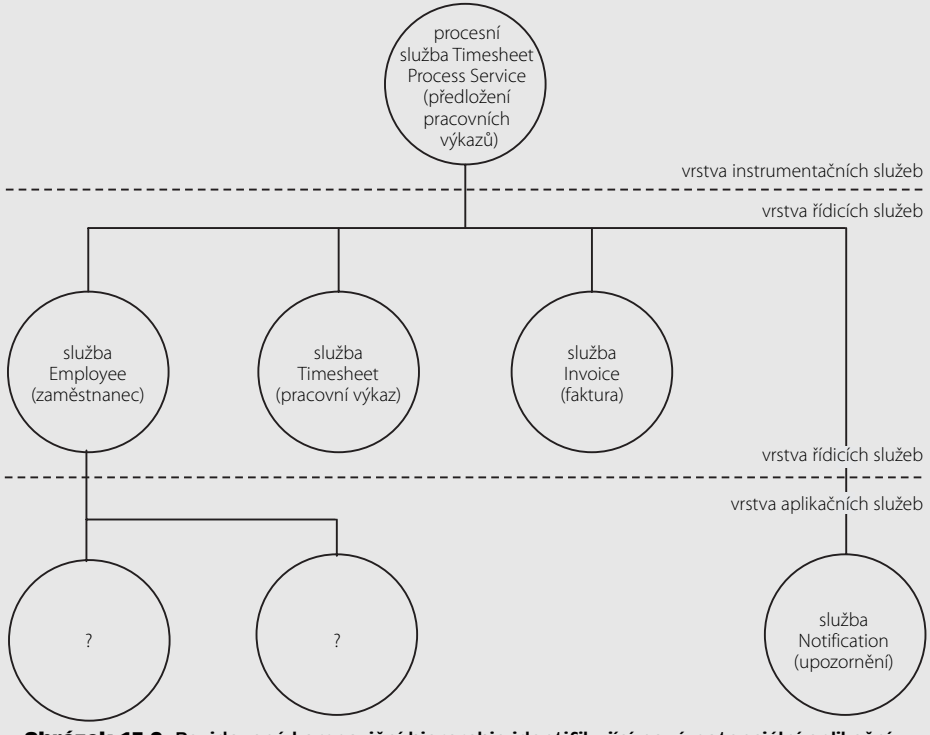
Objevili, že ačkoli toto schéma přesně vyjadřuje logickou datovou entitu, představuje strukturu dokumentu odvozenou od dvou odlišných fyzických úložišť. Následná analýza odhalila, že hodnota týdenního hodinového limitu je uložena v účetní databázi. Procesní požadavky pro operaci GetWeeklyHoursLimit jsou tedy zapsány takto:

Aplikační funkce na úrovni služby schopná vydat následující dotaz na účetní databázi: „Vrať týdenní hodinový limit zaměstnance pomocí ID zaměstnance jako jediného vyhledávacího kritéria."

Nyní bylo nutné prostudovat podrobnosti skrývající se za operací UpdateHistory (aktualizuj historii). Tentokrát je to poněkud snazší, neboť schéma EmployeeHistory.xsd je spojeno s jediným datovým zdrojem, kterým je repozitář správce lidských zdrojů s profily zaměstnanců. Pročtením dokumentace původní analýzy architekti identifikovali jednu část informace, kterou toto konkrétní řešení bude potřebovat pro aktualizaci tohoto repozitáře. Z tohoto důvodu jde definice procesního požadavku za bezprostřední požadavky řešení:

Aplikační funkce na úrovni služby schopná vydat pomocí ID zaměstnance jakožto jakožto jediného kritéria aktualizaci sloupce „komentář" tabulky s historií zaměstnance v databázi správce lidských zdrojů v profilech zaměstnanců.

Vypadá to, že řešení Předložení pracovních výkazů může potřebovat nové aplikační služby na podporu procesních požadavků služby Employee (zaměstnanec), jak ilustruje rozšířená kompozice na obrázku 15.9. Oba z těchto nově identifikovaných požadavků bude třeba podrobit procesu modelování služeb popsanému v kapitole 12.



Obrázek 15.9. Revidovaná kompoziční hierarchie identifikující nové potenciální aplikační služby

Na konec bylo odhaleno, že pro službu Employee (zaměstnanec) bude potřeba pouze jedna nová aplikační služba, která bude zabalovat správce lidských zdrojů, a jež může také pomoci službě Timesheet (pracovní výkaz). Kromě toho bylo zjištěno, že zpracování vyžadované službou Invoice (faktura) lze splnit pomocí stávající služby společnosti TLS Accounts Payable (závazky).

PŘÍPADOVÉ STUDIE (PROCESS RESULTS)

Níže je uvedena finální verze definice služby Employee (zaměstnanec) se zapracovanými změnami co se jmen elementů a všech předchozích revizí týče.

```
<definitions name="Employee"
   targetNamespace="http://www.xmltc.com/tls/employee/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:act="http://www.xmltc.com/tls/employee/schema/accounting/"
   xmlns:hr="http://www.xmltc.com/tls/employee/schema/hr/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.xmltc.com/tls/employee/wsdl/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/tls/employee/schema/">
      <xsd:import namespace=
         "http://www.xmltc.com/tls/employee/schema/accounting/"
         schemaLocation="Employee.xsd"/>
      <xsd:import namespace=
         "http://www.xmltc.com/tls/employee/schema/hr/"
         schemaLocation="EmployeeHistory.xsd"/>
      </xsd:schema>
</types>
<message name="getWeeklyHoursRequestMessage">
   <part name="RequestParameter"
      element="act:EmployeeHoursRequestType"/>
</message>
<message name="getWeeklyHoursResponseMessage">
   <part name="ResponseParameter"
      element="act:EmployeeHoursResponseType"/>
</message>
<message name="updateHistoryRequestMessage">
   <part name="RequestParameter"
      element="hr:EmployeeUpdateHistoryRequestType"/>
</message>
<message name="updateHistoryResponseMessage">
   <part name="ResponseParameter"
      element="hr:EmployeeUpdateHistoryResponseType"/>
</message>
<portType name="EmployeeInterface">
   <documentation>
      Operace GetEmployeeWeeklyHoursLimit (vezmi týdenní hodinový
      limit zaměstnance) používá ID zaměstnance
      pro načtení hodnoty WeeklyHoursLimit (týdenní hodinový limit).
      Operace UpdateEmployeeHistory (aktualizuj historii zaměstnance)
      používá ID zaměstnance pro aktualizaci hodnoty
      Comment (komentář) v rámci entity
      EmployeeHistory (historie zaměstnance).
```

```
      </documentation>
      <operation name="GetWeeklyHoursLimit">
         <input message="tns:getWeeklyHoursRequestMessage"/>
         <output message="tns:getWeeklyHoursResponseMessage"/>
      </operation>
      <operation name="UpdateHistory">
         <input message="tns:updateHistoryRequestMessage"/>
         <output message="tns:updateHistoryResponseMessage"/>
      </operation>
   </portType>
   ...
</definitions>
```

Výpis 15.8. Finální abstraktní definice služby

Poznámka

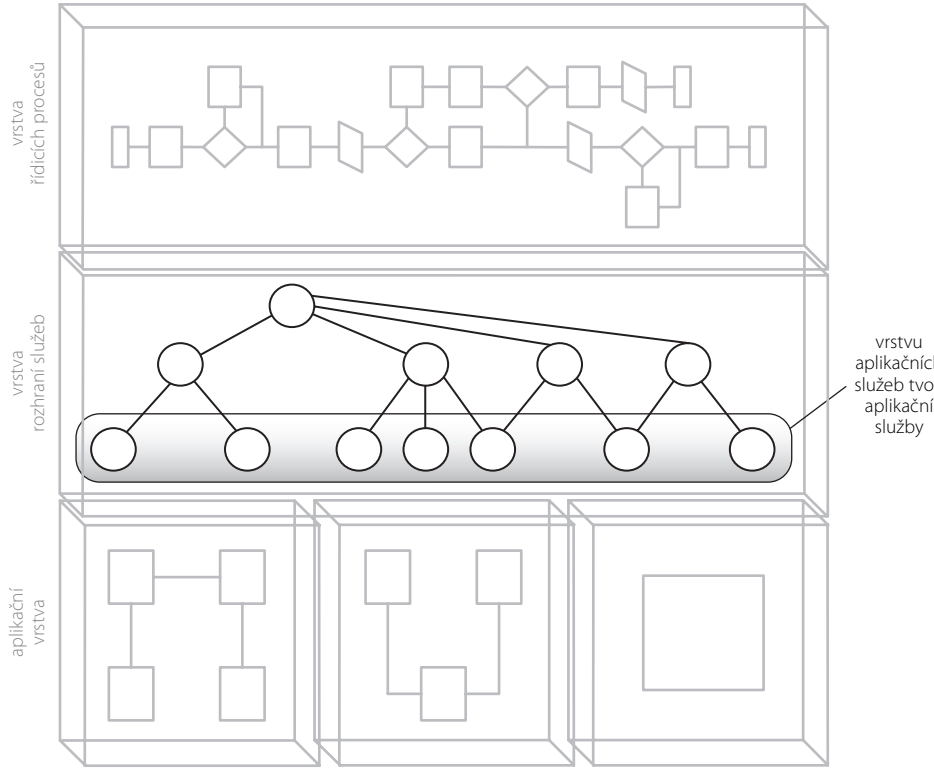
Výsledkem tohoto procesu je pouze abstraktní definice. Kompletní WSDL dokument včetně konkrétních detailů definice společně s importovanými XSD schématy lze stáhnout na adrese *www.serviceoriented.ws*.

SHRNUTÍ HLAVNÍCH POZNATKŮ

- Entitně zaměřené řídicí služby je nutné navrhovat tak, aby přesně reprezentovaly stávající podnikové entity, a přitom zůstaly agnostické vzhledem k řídicím procesům.
- Pro správnou výbavu entitně zaměřených řídicích služeb nezbytnou sadou generických operací může být nutné provést spekulativní analýzu.

Návrh aplikačních služeb (krok za krokem)

Aplikační služby jsou takovou tažnou silou SOA. Představují spodní podvrstvu složené vrstvy služeb (viz obrázek 15.10) odpovědnou za provádění jakýchkoli požadavků na zpracování od řídicí a instrumentační vrstvy.



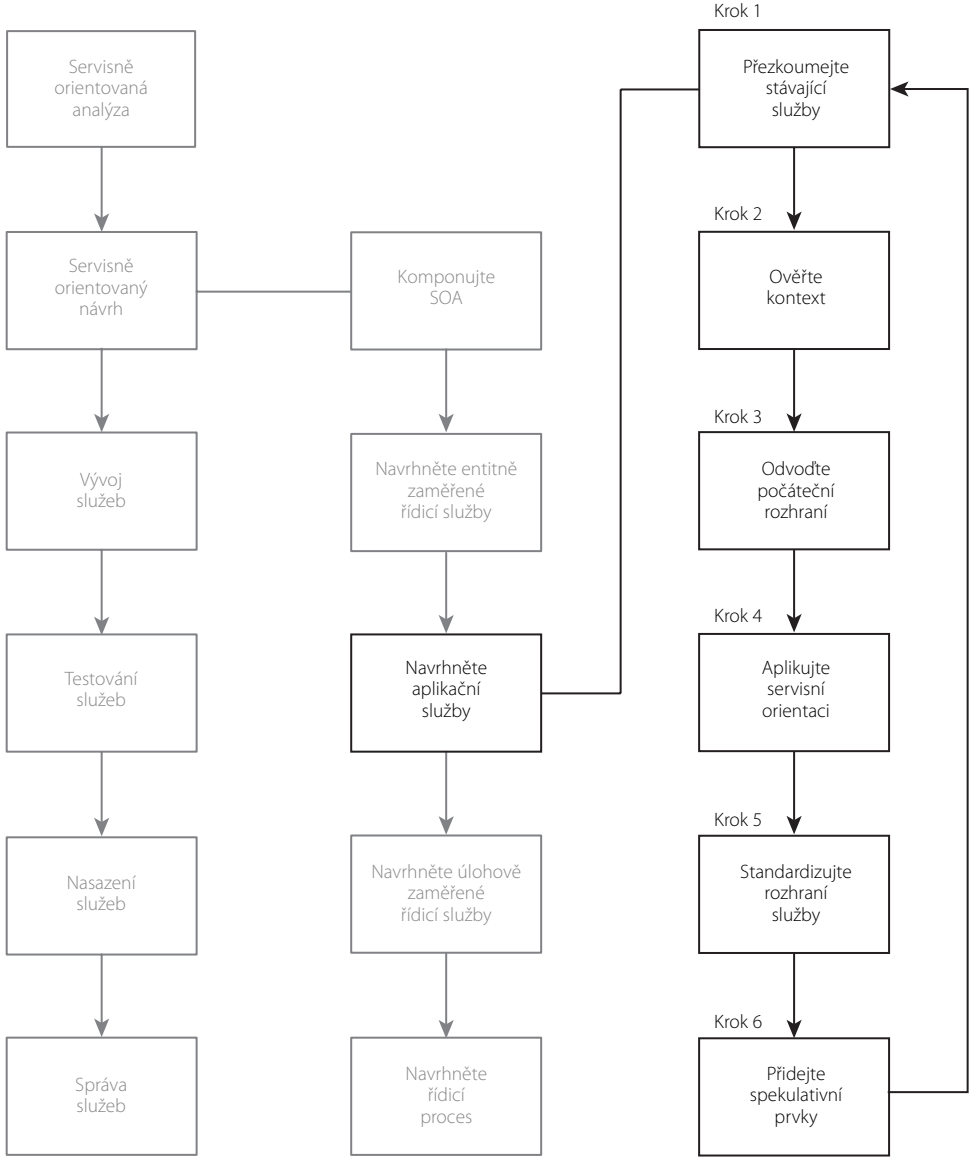
Obrázek 15.10. Aplikační služby tvoří v rámci vrstvy služeb spodní podvrstvu

Na rozdíl od služeb v řídicích vrstvách nevyžaduje návrh aplikačních služeb odbornou podnikovou analýzu. Vrstva aplikačních služeb je čistou, servisně orientovanou abstrakcí technických prostředí organizace, kterou dovedou nejlépe definovat ti, když nejvíce rozumí těmto prostředím.

Vzhledem k mnoha technologickým i praktickým činitelům, které je třeba vzít v potaz, může být návrh aplikačních služeb jeden z nejobtížnějších. Kromě toho může kvůli modernizaci či nahrazování technologií a sestavování či úpravě související aplikační logiky kontext ustavený těmito službami vyžadovat neustálé řešení.

Popis procesu

Obrázek 15.11 nabízí navrhovaný proces návrhu služeb pro vytvoření rozhraní aplikačních služeb. Všimněte si, že všechny odkazy na „aplikační služby“ v této i zbývajících kapitolách vyjadřují, že jde o znovupoužitelné pomocné aplikační služby.



Obrázek 15.11. Proces návrhu aplikačních služeb

Při pohledu na obrázek 15.11 je patrné, že tento proces s předchozím procesem entitně zaměřených řídicích služeb sdílí několik kroků. To je dáno tím, že jak aplikační, tak i entitně zaměřené služby tvoří znovupoužitelnou logiku služeb, a proto se opírají o nadřazené řadiče, které je skládají do úloh specifických pro konkrétní obchodní procesy.

Nicméně existují klíčové aspekty, v nichž se tyto dva procesy liší. Všimněte si například vymezení ověření kontextu seskupení operací do samostatného kroku. Ustavení kontextu pro aplikační služby je důležitou a mnohem méně zřejmou částí návrhu služeb.

Dále zde nejsou žádné kroky s definicí procesních požadavků. To je dáno především tím, že aplikační služby jsou odpovědné za implementaci procesních detailů nezbytných k provádění řídicí logiky jejich nadřazených řídicích služeb. To samozřejmě neznamená, že procesní požadavky aplikačních služeb neexistují. Existují, ovšem jen jako část návrhu základní aplikační logiky služby. To ovšem není součástí procesu, protože v této fázi navrhujeme pouze rozhraní služby.

Začněme tedy dávat dohromady rozhraní aplikačních služeb.

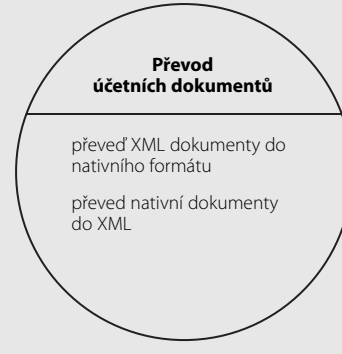
PŘÍPADOVÉ STUDIE

Nyní se přesuneme do prostředí společnosti RailCo, kde se zaměříme na návrh kandidáta aplikační služby Převod účetních dokumentů (Transform Accounting Documents), kterého jsme vymodelovali v kapitole 12 (viz obrázek 15.12).

Tento kandidát ustavuje „kontext pro transformaci dokumentů", což ospravedlňuje seskupení jeho dvou velmi podobných kandidátů operací:

- převeď XML dokumenty do nativního formátu,
- převed nativní dokumenty do XML.

Uvedené dva řádky informací tvoří základ, z něhož můžeme pomocí následujících kroků návrhového procesu odvodit fyzický návrh služby.



Obrázek 15.12. Kandidát služby Převod účetních dokumentů (Transform Accounting Documents)

Krok 1: Přezkoumejte stávající služby

U aplikačních služeb je více než u jakéhokoli jiného typu znovupoužitelných služeb důležité mít jistotu, že požadovaná funkcionalita daného kandidáta aplikační služby dosud v žádné formě neexistuje. Je tedy opravdu nezbytné přezkoumat stávající inventář aplikačních služeb, neobsahuje-li cokoliv, co by se podobalo chystanému návrhu.

Kromě toho je v této fázi vzhledem k obecné funkcionalitě nabízené těmito službami vhodné prošetřit, zda by nebylo možné požadované funkce koupit či pronajmout od dodavatelů třetích stran. Díky tomu, že aplikační služby by měly být navržené pro maximální znovupoužitelnost, mohou být webové služby třetích stran (které jsou obvykle postavené jako opětovně použitelné) skutečné vhodným řešením, pokud s nimi lze dosáhnout požadované úrovně kvality služeb.

PŘÍPADOVÉ STUDIE

Společnost RailCo dodává tuto službu jako součást řešení, jenž nahrazuje jejich původní hybridní služby Invoice Submission (předložení faktur) a Order Fulfillment (plnění objednávek). Jedinou další službou, jež existuje v rámci prostředí společnosti RailCo, je služba Předplatné TLS (TLS Subscription) používaná pro interakci s publikačními rozšířeními společnosti TLS. Tento krok je tedy hotový poměrně rychle, neboť je snadné ověřit, že funkcionalita plánovaná pro službu Převod účetních dokumentů (Transform Accounting Documents) nebude redundantní.

Krok 2: Ověřte kontext

Při provádění servisně orientované analýzy je přirozené zaměřit se na bezprostřední podnikové požadavky. Výsledkem pak ale je, že kandidáti aplikačních služeb vytvoření v této fázi často neberou v potaz kontexty ustavené stávajícími aplikačními službami.

Z tohoto důvodu je důležité, aby bylo přehodnoceno seskupení kandidátů operací navrhované kandidáty služeb a porovnáno s návrhy existujících aplikačních služeb. Po přehodnocení kontextu služby můžete shledat, že jedna či více operací ve skutečnosti náleží do jiných aplikačních služeb.

Poznámka

Tento krok nebyl v rámci návrhu entitně zaměřených řídicích služeb vyžadován, protože kontext entitně zaměřených služeb je předem definován odpovídajícími modely entit.

PŘÍPADOVÉ STUDIE

Přezkoumání jedné stávající služby společnosti RailCo a dalších služeb plánovaných jako část tohoto řešení potvrzuje, že seskupující kontext navrhovaný pro dva kandidáty operací kandidáta služby Převod účetních dokumentů (Transform Accounting Documents) je platný.

Krok 3: Odvoďte počáteční rozhraní služby

Analyzujte navrhované kandidáty operací služby a podle níže uvedených kroků definujte první návrh rozhraní služby.

1. Pomocí kandidáta aplikační služby jakožto primárního vstupu se ujistěte, že členění částí logiky reprezentované kandidáty operací jsou příslušným způsobem obecná a znovupoužitelná.
2. Zdokumentujte vstupní a výstupní hodnoty vyžadované pro zpracování každého kandidáta operace a pomocí konstruktů XSD schématu (jež v podstatě utvářejí WSDL konstrukt `types`) definujte strukturu zpráv.
3. Dokončete definici abstraktní služby přidáním oblasti `portType` či `interface` (společně s podřízenými konstrukty `operation`) a nezbytné konstrukty `message` obsahující elementy `part`, jež odkazují na příslušné typy schémat.

Všimněte si, že aplikační služby budou podobně jako generické jednotky procesní logiky používány různými typy řídicích služeb. Každá řídicí služba bude zpracovávat odlišné typy obchodních dokumentů (faktur, objednávek, pohledávek atd.). Aplikační služby tedy musejí být navrženy tak, aby mohly zpracovávat vícero typů dokumentů. V závislosti na povaze zpracovávané informace existuje několik možností návrhu.

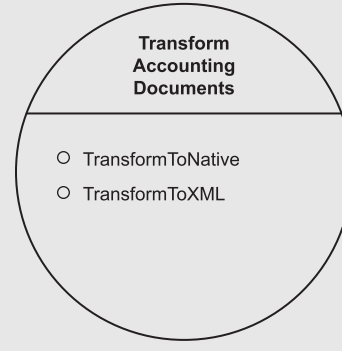
Příklady zahrnují:

- Vytvoření sady operací, které jsou generické, a přitom specifické pro daný dokument. Například místo jediné operace `Add` (přidat) byste mohli nabídnout samostatné operace `AddInvoice` (přidat fakturu), `AddPO` (přidat objednávku) a `AddClaim` (přidat pohledávku).
- Aplikační služby lze vybavit podporou SOAP příloh, které umožňují, aby generické operace vydávaly generické SOAP zprávy obsahující specifický obchodní dokument.

PŘÍPADOVÉ STUDIE

Společnost RailCo začala odvozením dvou názvů operací vyobrazených na obrázku 15.13.

Poté postoupila k definici konstruktu `types` v definici své služby pro formalizaci struktury zpráv. Nejdříve se pustila do zpráv požadavků a odpovědí operace `TransformToNative` (převeď na nativní).



Obrázek 15.13. První náčrt služby Transform Accounting Documents (převod účetních dokumentů)

```
<xsd:schema targetNamespace=
   "http://www.xmltc.com/railco/transform/schema/">
   <xsd:element name="TransformToNativeType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="SourcePath" type="xsd:string"/>
            <xsd:element name="DestinationPath" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="TransformToNativeReturnCodeType">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Code" type="xsd:integer"/>
```

```
            <xsd:element name="Message" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Výpis 15.9. Typy XSD schématu vyžadované operací TransformToNativ (převeď na nativní)

Po zhodnocení požadavků zpráv operace `TransformToXML` (převeď na XML) společnost RailCo objevila, že požadované typy jsou totožné. Zvažuje se úprava návrhu schématu pomocí sdílených složených typů, ale společnost RailCo je proti. Rozhoduje se pro vytvoření druhé sady elementů s redundantními složenými typy, protože má radši volnou ruku pro nezávislou změnu těchto typů v budoucnu.

```
<xsd:element name="TransformToXMLType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="SourcePath" type="xsd:string"/>
         <xsd:element name="DestinationPath" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="TransformToXMLReturnCodeType">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Code" type="xsd:integer"/>
         <xsd:element name="Message" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Výpis 15.10. Dodatečné typy pro operaci TransformToXML (převeď na XML)

Dále je obstaráním zbývajících konstruktů `message` a `portType` dokončena počáteční verze abstraktní definice služby pro službu Transform Account Documents (převod účetních dokumentů) společnosti RailCo.

```
<message name="transformToNativeRequestMessage">
   <part name="RequestParameter"
      element="trn:TransformToNativeType"/>
</message>
```

```
<message name="transformToNativeResponseMessage">
   <part name="ResponseParameter"
      element="trn:TransformToNativeReturnCodeType"/>
</message>

<message name="transformToXMLRequestMessage">
   <part name="RequestParameter"
      element="trn:TransformToXMLType"/>
</message>

<message name="transformToXMLResponseMessage">
   <part name="ResponseParameter"
      element="trn:TransformToXMLReturnCodeType"/>
</message>

<portType name="TransformInterface">
   <operation name="TransformToNative">
      <input message="tns:transformToNativeRequestMessage"/>
      <output message="tns:transformToNativeResponseMessage"/>
   </operation>
   <operation name="TransformToXML">
      <input message="tns:transformToXMLRequestMessage"/>
      <output message="tns:transformToXMLResponseMessage"/>
   </operation>
</portType>
```

Výpis 15.11. Konstrukty message a portType abstraktní definice služby Transform Account Documents (převod účetních dokumentů)

Poznámka

Společnost RailCo má od společnosti TLS nařízeno budovat služby, které vyhovují specifikaci WS-I Basic Profile. To vyžaduje, aby společnosti RailCo používala pro sestavování rozhraní služeb specifikaci jazyka WSDL 1.1. To jí ovšem vyhovuje, protože žádný z jejích middlewarů novější verzi jazyka WSDL stejně nepodporuje.

Krok 4: Aplikujte principy servisní orientace

Tento krok zdůrazňuje čtyři principy servisní orientace, které jsme si v kapitole 8 uvedli jako ty, jež nejsou ve skutečnosti nabízené platformou webových služeb (znovupoužitelnost, autonomie, bezstavovost a zjistitelnost služeb).

Znovupoužitelnosti jsme se věnovali v procesu návrhu služeb a přímo jej budeme řešit v kroku 5, kde se podíváme na to, jak udělat naše aplikační služby pro jejich potenciální žadatele co možná nejužitečnější. Nicméně stávající kandidáti operací by měli být také přezkoumáni, jsou-li navrženi generickým a opětovně použitelným způsobem.

Autonomie je primárním zájmem při návrhu aplikačních služeb. Musíme zajistit, aby základní aplikační logika odpovědná za provádění operací služby nekladla závislosti na danou službu nebo neměla sama závislosti. V tomto okamžiku nám údaje shromážděné v kroku 2 servisně orientované analýzy poskytují výchozí bod k prošetření povahy aplikační logiky, kterou každá operace služby potřebuje vyvolat. Krok 6 nabízí analýzu, která se věnuje této i dalším technologicky zaměřeným otázkám.

Bezstavovost může být pro aplikační služby obtížně dosažitelná. Vzhledem k tomu, že musejí spolupracovat s nejrůznějšími typy různých aplikačních platforem, jsou tyto služby vystavené vysoce nepředvídatelným implementačním prostředím. Dříve nebo později musejí aplikační služby čelit výzvám, jež kladou bezdůvodné či nekonzistentní výkonové požadavky (tím jsou známé především zastaralé původní systémy). Nejlepším způsobem k propagaci návrhu bezstavových aplikačních služeb je proto provádění počáteční analýzy v maximální možné míře. Znát předem požadavky na výkon vám umožní prošetřit alternativy před postoupením ke konkrétnímu návrhu.

Podobně jako v případě entitně zaměřených služeb, zjistitelnost může být důležitou součástí vyvíjené vrstvy aplikačních služeb. Pro záruku, že daný návrh se nekryje s logikou poskytovanou jinými aplikačními službami, je mechanismus zjistitelnosti velice užitečný. Stává se pak spíše infrastrukturním požadavkem, který lze plánovat jako součást implementace SOA. Nicméně pravidlo Dokumentujte služby pomocí metadat stále platí, neboť aplikační služby by měly být doplněny těmito metadaty v maximální možné míře.

PŘÍPADOVÉ STUDIE

Služba Transform Account Documents (převod účetních dokumentů) prochází revizí, aby se zajistilo, že se náležitým způsobem řídí principy servisní orientace.

Nejdříve je zhodnocena znovupoužitelnost dvou operací:

- TransformToNative (převeď na nativní),
- TransformToXML (převeď na XML).

Po určité diskusi ohledně toho, zda by tyto dvě operace měly být zkombinovány do jedné generické operace `Transform` (převeď), bylo rozhodnuto ponechat je tak, jak jsou. Preferuje se popisná povaha operací a společnost RailCo by si ráda ponechala možnost nezávislého budoucího vývoje každé z operací.

Dále jsou projednávány otázky autonomie a bezstavovosti. Autonomie není problém, což je dáno tím, že logika nezbytná k provádění transformačních funkcí je obsažena uvnitř základní aplikační logiky služby. Jinými slovy, neexistují zde žádné závislosti na dalších programech.

Bezstavovost by také neměla představovat žádný problém, protože tato služba bude odpovědná za své vlastní zpracování.

Nakonec je dohodnuto, že pro lepší podporu budoucí zjistitelnosti se definice služby doplní o dodatečná dokumentační metadata.

```
<portType name="TransformInterface">
   <documentation>
      Načte XML dokument a převede jej
      na nativní formát účetního dokumentu.
   </documentation>
   <operation name="TransformToNative">
      <input message="tns:transformToNativeRequestMessage"/>
      <output message="tns:transformToNativeResponseMessage"/>
   </operation>
   <documentation>
      Načte nativní účetní dokument a převede jej
      na XML-dokument.
   </documentation>
   <operation name="TransformToXML">
      <input message="tns:transformToXMLRequestMessage"/>
      <output message="tns:transformToXMLResponseMessage"/>
   </operation>
</portType>
```

Výpis 15.12. Konstrukt portType služby Transform Account Documents (převod účetních dokumentů) s doplňujícími dokumentačními metadaty

Krok 5: Standardizujte a dopilujte rozhraní služby

I když se role a účel aplikačních služeb od jiných typů služeb liší, je důležité, aby byly navrženy stejným základním způsobem. Toho docílíme zajištěním, že výsledná WSDL definice aplikační služby je založena na týchž standardech a konvencích, jež používají ostatní.

Následuje seznam doporučených akcí, které můžete provést k dosažení standardizovaného a usměrněného návrhu služby:

- Aplikujte jakékoli stávající návrhové standardy související s rozhraním služby (jejich seznam je k dispozici v pravidle na konci této kapitoly).
- Přezkoumejte jakékoli charakteristiky současné SOA, které mají vaše služby podporovat, a zhodnoťte, zda je možné zabudovat podporu pro tuto charakteristiku do návrhu služby.
- Můžete také začlenit pravidla a osvědčené postupy specifikace WS-I Basic Profile do jakékoli možné míry.

PŘÍPADOVÉ STUDIE

V důsledku činitelů braných v potaz v rámci tohoto kroku podstoupí služba určité změny. Po přezkoumání standardů pro pojmenování je rozhodnuto, že zvolené názvy pro operace jsou ve shodě se stávajícími konvencemi. Ovšem jméno samotné služby už není. Proto je služba přejmenována z „Transform Accounting Documents" (převod účetních dokumentů) na „Transform Accounting" (převod účetních).

Obrázek 15.14. Finální návrh služby Transform (převod)

Bylo shledáno jako nezbytné podporovat primární charakteristiky současné SOA. Společnost RailCo se již dříve rozhodla, že chce být v budoucnu schopna rozšiřovat svou SOA s minimálním úsilím vynaloženým na opětovný vývoj. Z tohoto důvodu by do každé její služby měla být v jakékoli uskutečnitelné míře zapracována charakteristika rozšiřitelnosti.

Po přezkoumání návrhu služby je pomocí následujících úprav dosaženo vyšší rozšiřitelnosti:

- Jméno služby se opět změní a tentokrát se zkrátí na „Transform" (převod).
- Operace TransformXMLToNative (převeď XML na nativní) a TransformNativeToXML (převeď nativní na XML) se přejmenují na něco více obecného (viz obrázek 15.14). Nová jména jsou ForAccountingImport (pro účetní import) a ForAccountingExport (pro účetní export). Tato změna jmen je poté promítnuta také do jmen elementů `element` a `message`.

```
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/railco/transform/schema/">
      <xsd:element name="ForImportType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="SourcePath" type="xsd:string"/>
               <xsd:element name="DestinationPath" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForImportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code" type="xsd:integer"/>
               <xsd:element name="Message" type="xsd:string"/>
            </xsd:sequence>
```

```
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForExportType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="SourcePath" type="xsd:string"/>
               <xsd:element name="DestinationPath" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForExportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code" type="xsd:integer"/>
               <xsd:element name="Message" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>
```

Výpis 15.13. Revidovaný konstrukt types

Uvedené úpravy udělají z této služby transformační nástroj, který zpočátku podporuje pouze transformační funkce pro účetní systém, ale jejž lze rozšířit o další funkce související s transformací.

Krok 6: Vybavte kandidáty služeb spekulativními prvky

Pokud chcete vytvořit vysoce znovupoužitelné aplikační služby, můžete využít této příležitosti pro přidání prvků k jejich návrhu. Tyto nové prvky mohou ovlivnit stávající operace nebo mohou vyústit v přidání operací nových. Pro aplikační služby se spekulativní rozšíření točí kolem typu zpracování, jenž spadá do kontextu dané služby.

Samozřejmě že před vlastním přidáním spekulativních rozšíření k aplikační službě byste měli opakovat krok 1 pro ověření, že žádná z nových operací neexistuje v rámci jiných služeb. Kromě toho při přidávání nových rozšíření je nutné opakovat ještě kroky 2 až 4, aby bylo zajištěno, že jsou náležitě standardizované a navržené v souladu s dosud navrženým rozhraním služby.

PŘÍPADOVÉ STUDIE

Ačkoli by společnost RailCo ráda budovala vysoce znovupoužitelné aplikační služby, nemůže si to v tuto chvíli dovolit. Vzhledem k tomu, že službu Transform (převod) má navrženou dostatečně obecně a znovupoužitelně, aby ji bylo možné použít v procesech Předložení faktur a Plnění objednávek, je si společnost RailCo jistá, že její bezprostřední požadavky jsou splněny. A navíc díky zdůrazňování rozšiřitelnosti cítí, že pokud se v budoucnu objeví příležitosti k opětovnému použití této služby, může pokračovat v jejím dalším vývoji.

Krok 7: Identifikujte technická omezení

V této chvíli máme vytvořené ideální rozhraní služby, které se ovšem nachází ve „vakuu". Na rozdíl od řídicích služeb potřebujeme u aplikačních služeb vzít v potaz nízkoúrovňové činitele z reálného světa.

Nyní tedy potřebujeme blíže prostudovat a zdokumentovat procesní požadavky každé operace služby. Nejdříve pro každou operaci napíšeme seznam procesních funkcí nezbytných pro provedení dané operace. Poté pro každou položku na seznamu přesně zjistíme, jak bude prováděné zpracování funkce ve stávajícím technickém prostředí.

Mezi typy podrobností, které hledáme, patří:

- Fyzické spojovací body konkrétní funkce. (Jinými slovy, jaké komponenty je nutné vyvolat, jaké funkce aplikačního rozhraní zavolat, nebo které adaptéry aktivovat.)
- Bezpečnostní omezení související s jakoukoli částí zpracování.
- Doba odpovědi každé procesní funkce.
- Dostupnost základního systému provádějícího procesní funkce.
- Faktory prostředí související s místem nasazení služby.
- Technická omezení základní aplikační logiky (zejména tehdy, když se exponují původní systémy).
- Administrační požadavky kladené službou.
- Potenciální SLA požadavky.

Po shromáždění charakteristik jednotlivých procesních funkcí je třeba se na ně dívat jako na celek. Například je nutné sečíst jednotlivé doby odpovědi pro výpočet odhadu celkové doby provádění operace. Výsledkem této studie je obvykle série omezení a limitací kladených technickým prostředím na rozhraní naší služby. V některých případech budou restrikce tak tvrdé, že bude nutné danou operaci výrazným způsobem rozšířit.

Všimněte si, že při přechodu organizace směrem k celopodnikové architektuře SOA existuje sklon, aby vše bylo servisně orientované. Je však důležité identifikovat, které procesní požadavky nemohou být technologickou sadou webových služeb splněny. Asi nedává smysl exponovat některé části základní aplikační logiky původního systému jako webové služby.

Ať už tak, či onak, stojí za připomenutí, že i když se tato kniha zaměřuje na tvorbu služeb jako webových služeb, SOA je ve skutečnosti implementačně neutrální architektonický model a servisní orientace je implementačně neutrální návrhové paradigma. Stávající formy aplikační logiky nedostupné prostřednictvím webových služeb lze i tak modelovat jako služby. To je zvláště významné pro aplikační služby, kde exponování aplikační logiky přes webovou služ-

bu nemusí vždy být správné rozhodnutí. Například fasádní komponenty jsou často vytvářeny pro zapouzdření funkcionality z různých zdrojů, a pro následné exponování odlišného kontextu představujícího sadu znovupoužitelných funkcí. To pak vede ke skutečné službě, která může být v budoucnu i nadále vyjádřena přes webovou službu.

PŘÍPADOVÉ STUDIE

Jediná otázka, která vyvstane při hodnocení technických omezení služby Transform (převod), je potenciální překážka ve výkonu. Přezkoumání stávajících metrik ukazuje, že proces převodu může být při zpracování dokumentů faktury nebo objednávky s více než dvaceti položkami řádků časově náročný. Vzhledem k tomu, že služba bude provádět své vlastní zpracování, očekává se, že její instance bude spjata s průběhem transformačního procesu. Větší dokumenty tedy pravděpodobně způsobí znatelné časové prodlevy.

Toto odhalení jde proti našemu původnímu předpokladu, že udržovat bezstavovost by neměl být žádný problém. Nicméně, vzhledem k tomu, že společnost RailCo neočekává vysokou míru užívání služby, a protože transformační perioda nemá skutečný dopad na efektivitu zpracování jejích interních procesů týkajících se faktur a objednávek, je rozhodnuto, že návrh služby bude ponechán beze změn.

PŘÍPADOVÉ STUDIE (PROCESS RESULTS)

Následuje finální verze definice služby Transform (převod) se zapracovanými změnami názvů elementů a všech předchozích revizí.

```
<definitions name="Transform"
   targetNamespace="http://www.xmltc.com/railco/transform/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.xmltc.com/railco/transform/wsdl/"
   xmlns:trn="http://www.xmltc.com/railco/transform/schema/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <types>
      <xsd:schema targetNamespace=
         "http://www.xmltc.com/railco/transform/schema/">
         <xsd:element name="ForImportType">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="SourcePath" type="xsd:string"/>
                  <xsd:element name="DestinationPath" type="xsd:string"/>
               </xsd:sequence>
            </xsd:complexType>
```

```
      </xsd:element>
      <xsd:element name="ForImportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code" type="xsd:integer"/>
               <xsd:element name="Message" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForExportType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="SourcePath" type="xsd:string"/>
               <xsd:element name="DestinationPath" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="ForExportReturnCodeType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="Code" type="xsd:integer"/>
               <xsd:element name="Message" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>

<message name="ForAccountingImportRequestMessage">
   <part name="RequestParameter"
      element="trn:ForImportType"/>
</message>
<message name="ForAccountingImportResponseMessage">
   <part name="ResponseParameter"
      element="trn:ForImportReturnCodeType"/>
</message>
<message name="ForAccountingExportRequestMessage">
```

```
      <part name="RequestParameter"
         element="trn:ForExportType"/>
   </message>
   <message name="ForAccountingExportResponseMessage">
      <part name="ResponseParameter"
         element="trn:ForExportReturnCodeType"/>
   </message>

   <portType name="TransformInterface">
      <documentation>
         Operace ForAccountingImport načte XML dokument
         a převede jej do nativního formátu účetního dokumentu.
         Operace ForAccountingExport načte nativní účetní dokument
         a převede jej na XML dokument.
      </documentation>
      <operation name="ForAccountingImport">
         <input message="tns:ForAccountingImportRequestMessage"/>
         <output message="tns:ForAccountingImportResponseMessage"/>
      </operation>
      <operation name="ForAccountingExport">
         <input message="tns:ForAccountingExportRequestMessage"/>
         <output message="tns:ForAccountingExportResponseMessage"/>
      </operation>
   </portType>
   ...
</definitions>
```

Výpis 15.14. Výsledná abstraktní definice služby

Poznámka

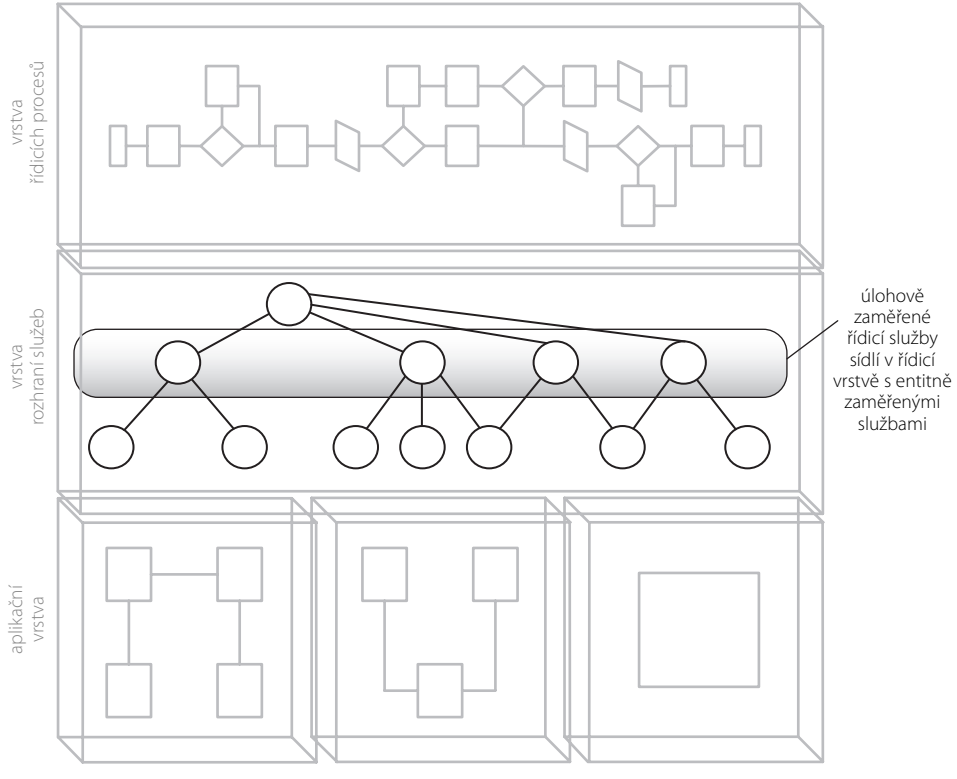
Výsledkem tohoto procesu je pouze abstraktní definice. Kompletní WSDL dokument včetně konkrétních detailů definice lze stáhnout na adrese *www.serviceoriented.ws*.

SHRNUTÍ HLAVNÍCH POZNATKŮ

- Aplikační služby je nutné navrhovat agnosticky vzhledem k řešení a implementovat model pomocných služeb, aby bylo možné maximalizovat znovupoužitelnost.
- Pro návrh rozhraní služby, jež exponuje přiměřeně generické a znovupoužitelné operace, může být vyžadována spekulativní analýza.

Návrh úlohově zaměřených řídicích služeb (krok za krokem)

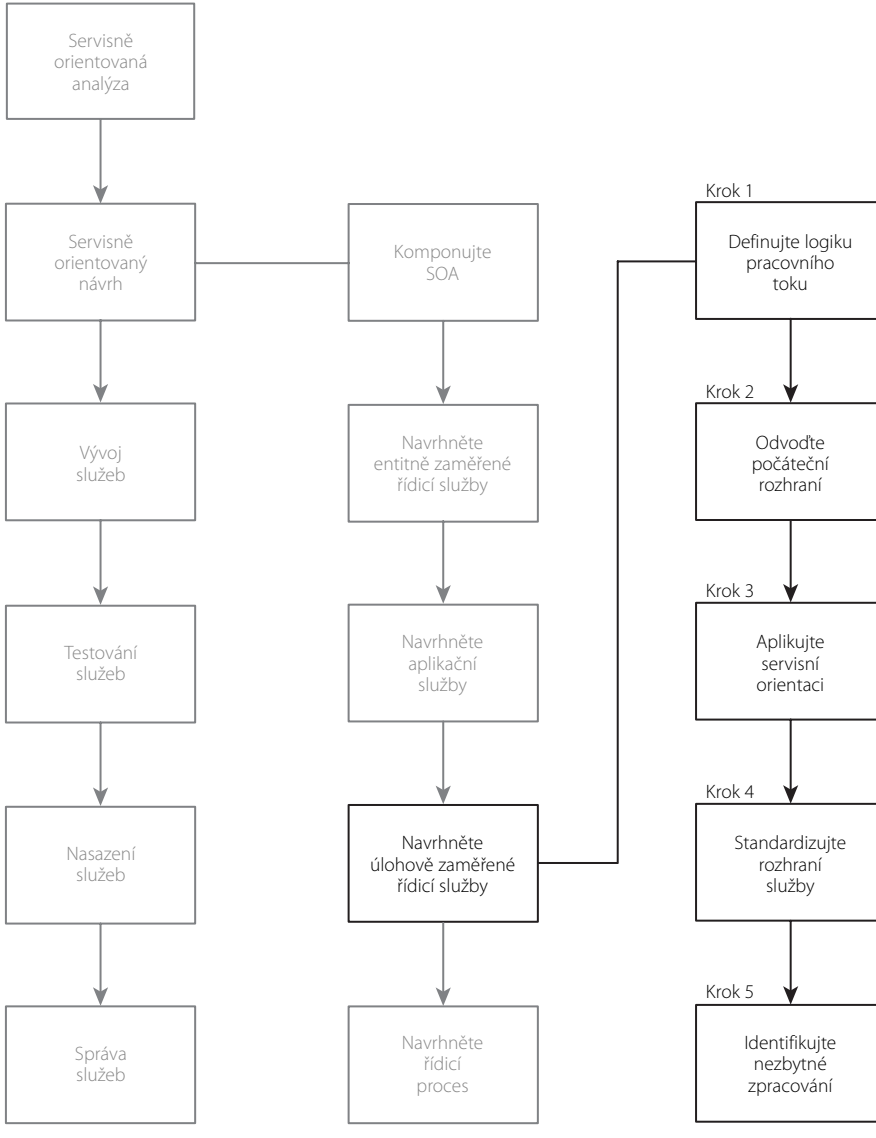
Procesy pro návrh úlohově zaměřených služeb obvykle vyžadují méně úsilí, než předchozí dva návrhové procesy, což je dáno jednoduše tím, že znovupoužitelnost není primárním činitelem. Proto se zde řeší pouze kandidáti operací služeb identifikovaní v rámci procesu modelování služeb.



Obrázek 15.15. Úlohově zaměřené řídicí služby mohou tvořit vrstvu řídicích služeb společně s entitně zaměřenými sousedy

Popis procesu

Jak je patrné z obrázku 15.16, začíná tento proces novým druhem kroku, v rámci něhož je načrtnut plán logiky pracovního toku. To je dáno tím, že u úlohově zaměřených řídicích služeb se očekává, že obsahují a řídí části řídicího procesu.



Obrázek 15.16. Proces návrhu úlohově zaměřených řídicích služeb

Všimněte si, že zde není žádný krok vyzývající k rozšíření sady funkcí v návrhu služeb definovaných ve fázi modelování služeb. Jak jsme se zmínili již dříve, není poskytnutí generického a znovupoužitelného rozhraní pro úlohově zaměřené služby prioritní.

Je čas pustit se do návrhu služeb.

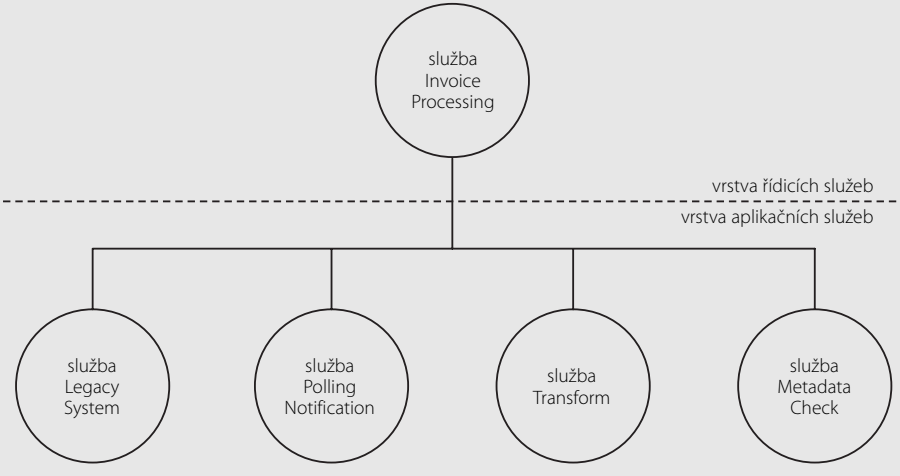
PŘÍPADOVÉ STUDIE

Při procesu modelování služeb společnosti RailCo byla identifikována potřeba úlohově zaměřené řídicí služby pro řízení zpracování faktur produkovaných původním účetním systémem. Výsledkem je kandidát služby Zpracování faktur zachycený na obrázku 15.17.

Obrázek 15.17. Kandidát služby Zpracování faktur

Na první pohled to vypadá, jako by tato služba nic moc neobsahovala. To však pro menší úlohově zaměřené služby nebývá výjimkou. Kandidát služby bez kandidátů operací jednoduše znamená, že daná služba je čistým řadičem, určeným výhradně ke koordinaci kompozice služeb. Dále to znamená, že společnost RailCo bude muset během tohoto procesu návrhu definovat rozhraní své služby zcela od nuly.

Naštěstí je zde výchozí bod, který poskytuje model kompozice (viz obrázek 15.18) vytvořený během kroku 6 procesu modelování služeb v kapitole 12.



Obrázek 15.18. Kompozice služby Invoice Processing (zpracování faktur)

Vypadá to, že služba Invoice Processing (zpracování faktur) bude ve skutečnosti docela zaneprázdněná, neboť musí ke zpracování jediného předložení faktury složit až čtyři samostatné služby.

Krok 1: Definujte logiku pracovního toku

Úlohově zaměřené služby budou obvykle obsahovat vloženou logiku pracovního toku používanou pro koordinaci podkladové kompozice služeb. Naším prvním krokem je proto definice této logiky pro každou možný scénář interakce, který si dokážeme představit. Pokud jste si v rámci procesu modelování služeb v kapitole 12 provedli cvičení v kroku *Identifikujte kandidátské kompozice služeb*, pak již budete mít detaily předběžné kompozice zdokumentované.

Protože navrhujeme úlohově zaměřené řídicí služby po dokončení návrhů entitně zaměřených a aplikačních služeb, musíme tyto dokumenty se scénáři revidovat a udělat z nich konkrétní modely interakce služeb.

K uskutečnění tohoto kroku lze použít různé tradiční modelovací přístupy (v příkladech naší případové studie použijeme sekvenční diagramy). Smyslem tohoto cvičení je zdokumentovat každý myslitelný průběh spouštění, včetně všech výjimečných stavů. Výsledné diagramy budou také užitečným vstupem pro následné testovací případy.

Poznámka

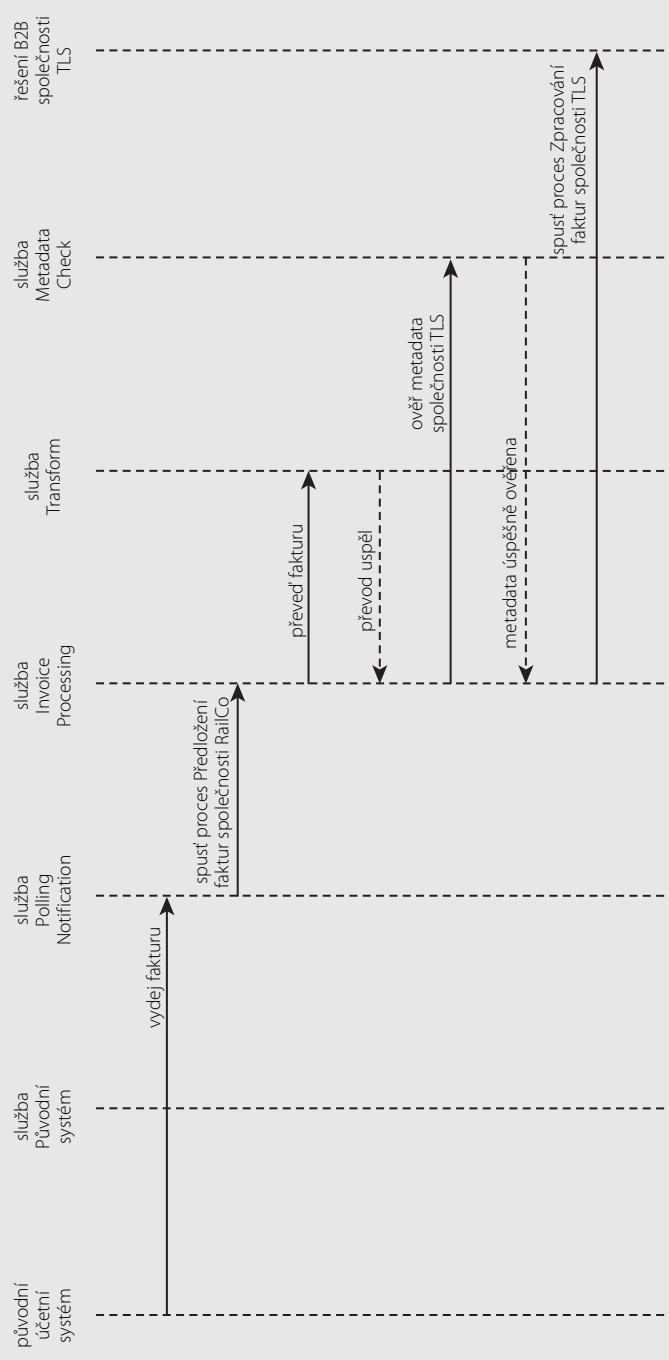
Logika pracovního toku nesídlí v rozhraní služby, které v tomto procesu navrhujeme. Logiku pracovního toku definujeme kvůli extrakci výměny zpráv, do které bude tato služba zapojena. To nám poskytne informace, jež nám pomohou definovat typy, operace a formáty zpráv.

PŘÍPADOVÉ STUDIE

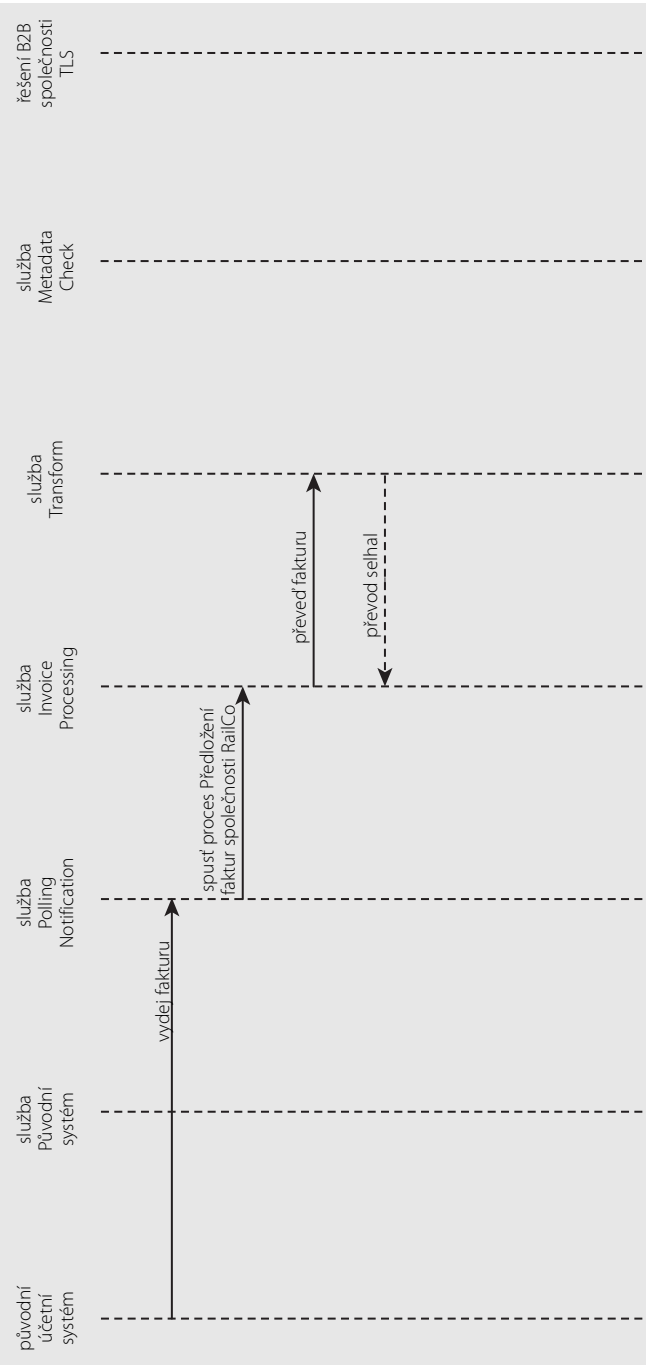
Společnost RailCo vytvořila sekvenční diagramy pro všechny představitelné scénáře interakcí zahrnující službu Invoice Processing (zpracování faktur). Podívejme se nyní na tyto dva diagramy.

Obrázek 15.19 znázorňuje podmínky a výměny zprávy nezbytné k úspěšnému dokončení předložení faktur.

Obrázek 15.20 ukazuje, jak chybový stav zastaví celý proces. V tomto případě vrací služba Transform (převod) nějakou chybu, která vznikla třeba kvůli přijetí neplatného dokumentu.



Obrázek 15.19. Úspěšné dokončení procesu Předložení faktur



Obrázek 15.20. Chybový stav způsobený chybou během převodu

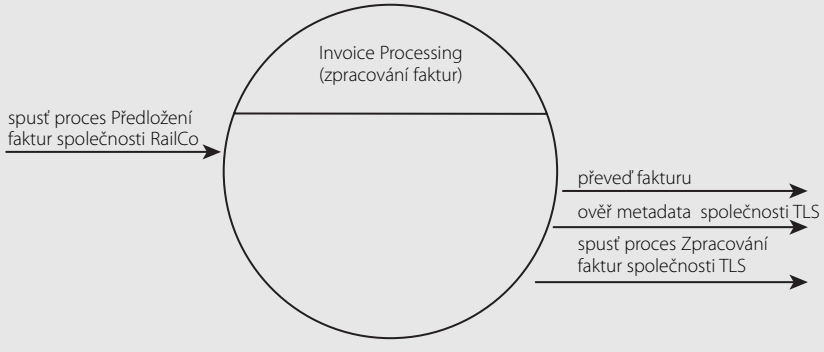
Krok 2: Odvoďte rozhraní služby

Pro sestavení počátečního rozhraní služby postupujte podle následujících kroků:

1. Pomocí kandidátů operací aplikační služby odvoďte sadu odpovídajících operací.
2. Zdroj, od něhož odvodíme rozhraní naší služby, obsahuje nyní na rozdíl od předchozích návrhových procesů také sekvenční diagramy a logiku pracovního toku, kterou jsme zdokumentovali v kroku 1. Tyto informace nám dají dobrou představu o tom, jaké dodatečné operace může naše úlohově zaměřená služba vyžadovat.
3. Zdokumentujte vstupní a výstupní hodnoty nezbytné pro zpracování každé operace a naplňte sekci `types` typy XSD schématu potřebnými k jejich zpracování.
4. Vytvořením oblasti `portType` (či `interface`) a vložením identifikovaných konstruktů `operation` sestavte WSDL definici. Poté přidejte nezbytné konstrukty `message` obsahující elementy part, jež odkazují na příslušné typy schématu.

PŘÍPADOVÉ STUDIE

Vzhledem k tomu, že nám kandidát služby neposkytl žádné kandidáty operací, obrací se společnost RailCo pro odvození sady akcí, které má daná služba provádět, k vytvořeným sekvenčním diagramům (viz obrázek 15.21):



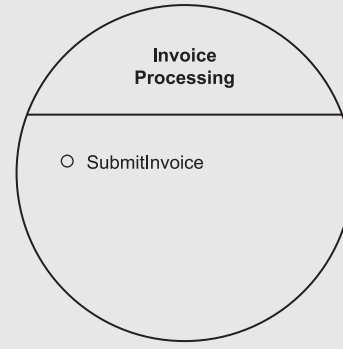
Obrázek 15.21. Identifikované dotazy a odpovědi pro službu Invoice Processing (zpracování faktur)

- **Spusť proces Předložení faktur společnosti RailCo – přijme hlášení odeslané službou Upozornění na aktualizace, která spouští proces Předložení faktur společnosti RailCo.**
- **Převeď fakturu – vydá dotaz na službu Transform (převod) pro načtení dokumentu faktury ze síťové složky a převede ji do XML.**
- **Ověř metadata společnosti TLS – vydá požadavek na službu Kontrola metadat pro určení, zda je čas ověřit metadata (a poté provede jejich ověření, je-li to požadováno).**
- **Spusť proces Zpracování faktur společnosti TLS – předá dokument faktury společnosti TLS, která zahájí samostatný proces zpracování faktur společnosti TLS.**

Z těchto akcí vyžadují poslední tři, aby se služba chovala jako žadatel pro zahájení výměny zpráv s dalšími službami. Tyto akce tedy budou implementovány jako součást podkladové řídicí logiky služby.

Akce „spusť proces Předložení faktur společnosti RailCo" je spouštěna službou Upozorňování na aktualizace, což znamená, že služba Zpracování faktur přijme požadavek, zatímco funguje jako poskytovatel služby. Tato akce tedy musí být vyjádřena v rozhraní služby.

Nejdříve je třeba zvážit vhodné pojmenování, jak ukazuje obrázek 15.22.



Obrázek 15.22. Služba Invoice Processing (zpracování faktur) s novou operací

Společnost RailCo začíná definováním požadavků na výměnu dat pro tuto jedinou operaci. Návrh rozhraní služby potřeboval spolupracovat se službou Upozorňování na aktualizace, k čemuž vyžaduje hodnotu ID a cestu k umístění souboru s dokumentem faktury. (To je dáno tím, že služba Upozorňování na aktualizace ve skutečnosti fyzicky nenačítá a nepředává dokumenty, ale jednoduše upozorní další služby na příchod specifického typu souboru do konkrétní složky.)

Společnost RailCo začala tvorbou předběžných konstruktů `types` s následujícím složeným typem `complexType`, který odpovídá požadavkům na parametry.

```
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/railco/invoiceservice/schema/">
      <xsd:element name="SubmitInvoiceType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="ContextID" type="xsd:integer"/>
               <xsd:element name="InvoiceLocation" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>
```

Výpis 15.15. Konstrukt complexType navržený pro přijetí dvou parametrů od služby Upozorňování na aktualizace

Dále v rámci konstruktů `portType` a `message` definuje operaci a s ní spojené zprávy:

```
<message name="receiveSubmitInvoiceMessage">
   <part name="RequestParameter"
      element="invs:SubmitInvoiceType"/>
</message>
<portType name="InvoiceProcessingInterface">
   <operation name="SubmitInvoice">
      <input message="tns:receiveSubmitInvoiceMessage"/>
   </operation>
</portType>
```

Výpis 15.16. Zbývající části abstraktní definice služby Invoice Processing (zpracování faktur) ustavují operaci s pouze jedinou vstupní zprávou

Všimněte si, že díky tomu, že zpráva posílaná službou Upozorňování na aktualizace je založená na jednocestném vzoru MEP, obsahuje konstrukt `operation` s názvem SubmitInvoice (předlož fakturu) jeden element zprávy `input` a žádné elementy `output`.

Krok 3: Aplikujte principy servisní orientace

Před tím, než se pustíme hlouběji do návrhu našich služeb, je prospěšné znovu se podívat na čtyři principy servisní orientace, kterým jsme se věnovali v kapitole 8, a jež nám webové služby nenabízejí automaticky (znovupoužitelnost, autonomie, bezstavovost a zjistitelnost služeb).

Příležitosti ke znovupoužití úlohově zaměřených služeb jsou vzácnější než u entitně zaměřených a aplikačních služeb. To je dáno tím, že úlohově zaměřené služby představují část logiky pracovního toku specifickou pro určitý obchodní proces. I přesto lze však znovupoužitelnosti dosáhnout. Tento problém řeší modelovací pravidla *Vezměte v úvahu potenciální meziprocesní znovupoužitelnost zapouzdřované logiky* a *Uvažte potenciální vnitroprocesní znovupoužitelnost zapouzdřované logiky* v kapitole 12, která lze aplikovat i na tento proces.

Vzhledem k tomu, že úlohově zaměřené služby většinou fungují jako řídicí služby v kompozici, bývá jejich autonomie obecně závislá na autonomii podřízených služeb. Údržba konzistentního autonomního stavu tak může být poměrně obtížná.

Úlohově zaměřené služby obsahují logiku pracovního toku, jež může klást procesní závislosti na kompozice služeb. To může vést k nutnosti zavedení správy stavu. Nicméně použití SOAP zpráv v dokumentovém stylu umožňuje službě přenést perzistenci některé nebo všech těchto stavových informací do samotné zprávy.

Pro služby je vždy užitečné, aby byly zjistitelné, v případě úlohově zaměřených služeb však ne s takovou naléhavostí jako u ostatních, obecnějším způsobem opětovně použitelných služeb. Bez ohledu na to mohou i úlohově orientované služby být znovupoužitelné a jejich existence by měla být známa i ostatním. Proto lze také na ně vztáhnout pravidlo *Dokumentujte služby pomocí metadat* na konci této kapitoly.

PŘÍPADOVÉ STUDIE

Po službě Invoice Processing (zpracování faktur) není požadováno, aby byla znovupoužitelná a autonomie a bezstavovost také nejsou považovány za bezprostřední zájmy. Podobně jako v případě dříve navržené služby Transform (převod) společnosti RailCo, je i návrh této služby doplněn o dodatečná dokumentační metadata pro podporu zjistitelnosti.

```
<portType name="InvoiceProcessingInterface">
   <documentation>
      Zahájí proces Předložení faktur.
   </documentation>
   <operation name="SubmitInvoice">
      <input message="tns:receiveSubmitInvoiceMessage"/>
   </operation>
</portType>
```

Výpis 15.17. Konstrukt portType s dodatečným elementem documentation

Krok 4: Standardizujte a vypilujte rozhraní služby

I když úlohově zaměřené řídicí služby mají sklon ke kreativnějším názvům operací, je třeba aplikovat stávající konvence. Níže je uveden standardní seznam doporučených akcí, které můžete provést k dosažení standardizovaného a řádně usměrněného návrhu služby:

- Začleňte existující návrhové standardy a pravidla. (Sadu doporučených pravidel nabízí poslední část této kapitoly.)
- Zajistěte, aby jakékoli zvolené charakteristiky současné SOA byly plně podporovány návrhem rozhraní služby.
- Vezměte v potaz standardy a osvědčené postupy specifikace WS-I Basic Profile.

S ohledem na návrhové standardy související se členěním operací může být vyžadována určitá shovívavost pro správné zpracování posloupnosti logiky pracovního toku dané služby. Úlohově zaměřené služby mohou také těžit z opětovného použití stávajících WSDL modulů, konkrétně z definic XSD schémat.

PŘÍPADOVÉ STUDIE

Podpora charakteristik rozšiřitelnosti je pro společnost RailCo klíčová, neboť si není jistá, do jaké míry se bude její SOA v průběhu času rozrůstat a vyvíjet. Při přezkoumávání definice rozhraní služby Invoice Processing (zpracování faktur) si uvědomila, že toto rozhraní přizpůsobují jedinému žadateli: službě Upozornění na aktualizace. Dá se předvídat, že procesní logiku zapouzdřenou v této službě by v budoucnu mohlo být nutné vyvolat i jiným způsobem.

Například:

- Změna celého procesu předložení faktur může vyžadovat, aby tato služba posílala předtransformovanou XML verzi aktuálního dokumentu faktury.

- Pokud se budou do technického prostředí společnosti RailCo stále přidávat nové služby, bylo by pro tuto službu prospěšné přijímat dokument faktury, aby se mohla účastnit větších kompozic služeb.

Pro vyřešení těchto požadavků na rozšiřitelnost provedla společnost RailCo úpravu značkovacího kódu schématu v rámci konstruktu `types`. Díky tomu, že XSD schéma představující dokument faktury již existuje (jako výsledek budování podkladové procesní logiky pro službu Transform), bylo rozhodnuto jej začlenit do této WSDL definice.

Aby nedošlo k vytváření redundantního značkovacího kódu, se schéma importuje do konstruktu `types`. Původní element `complexType` se poté rozšíří tak, aby obsahoval nový element představující kořenový element dokumentu faktury definovaný v souboru Invoice.xsd. Žadatelé služby nyní musejí poslat buď umístění dokumentu, nebo samotný dokument.

```
<types>
   <xsd:schema targetNamespace=
      "http://www.xmltc.com/railco/invoiceservice/schema/">
      <xsd:import namespace=
         "http://www.xmltc.com/railco/invoice/schema/"
         schemaLocation="Invoice.xsd"/>
      <xsd:element name="SubmitInvoiceType">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="ContextID" type="xsd:integer"/>
               <xsd:element name="InvoiceLocation" type="xsd:string"/>
               <xsd:element name="InvoiceDocument" type="inv:InvoiceType"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>
```

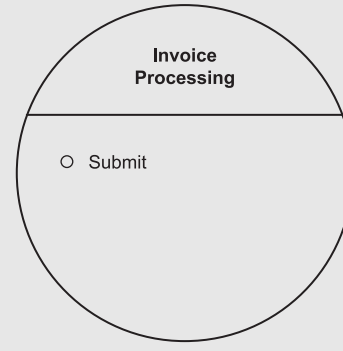
Výpis 15.18. Revidovaný konstrukt types z definice služby Invoice Processing (zpracování faktur)

Skutečnost, že došlo k úpravě přípustných vstupních hodnot, by se měla odrazit v obsahu elementu `documentation`:

```
<documentation>
   Zahájí proces Předložení faktur.
   Vyžaduje buď umístění dokumentu faktury,
   nebo samotný dokument.
</documentation>
```

Výpis 15.19. Obsah elementu documentation byl také změněn

Během tohoto kroku procesu návrhu služeb byla provedena ještě jedna změna. Bylo znovu uvažováno nad původním jménem operace „SubmitInvoice" (předlož fakturu). Podle interní konvence pro pojmenování bylo toto jméno zkráceno na „Submit" (předlož). Protože jméno služby již odráží fakt, že její kontext se točí kolem zpracování dokumentu faktury, není nutné opakovat slovo „invoice" (faktura) ještě v názvech operací (viz obrázek 15.23).



Obrázek 15.23. Služba Invoice Processing (zpracování faktur) s revidovaným jménem operace

Tato změna se promítne do obou konstruktů `portType` a `message`:

```
<message name="receiveSubmitMessage">
   <part name="RequestParameter"
      element="invs:SubmitInvoiceType"/>
</message>
<portType name="InvoiceProcessingInterface">
   <documentation>
      Zahájí proces Předložení faktur.
      Vyžaduje buď umístění dokumentu faktury,
      nebo samotný dokument.
   </documentation>
   <operation name="Submit">
      <input message="tns:receiveSubmitMessage"/>
   </operation>
</portType>
```

Výpis 15.20. Revidované konstrukty message a portType

Krok 5: Identifikujte nezbytné zpracování

Pro provádění svého dílu procesní logiky řešení mohou úlohově zaměřené služby komponovat aplikační, entitně zaměřené a další úlohově zaměřené řídicí služby. Proto vyžaduje implementace rozhraní úlohově zaměřené služby, aby byly k dispozici jakékoli potřebné základní vrstvy služeb podporující zpracování požadavků pro její operace.

Protože se jedná o poslední z našich tří procesů pro návrh služeb, je třeba identifikovat všechny požadované podpůrné služby. Mohou se skládat ze služeb, které již existují, anebo ze služeb, jež jsme právě navrhli během předchozího procesu návrhu aplikačních služeb. Návrh procesní logiky v rámci úlohově zaměřených řídicích služeb může také odhalit potřebu dodatečných aplikačních služeb, o kterých se dosud neuvažovalo.

PŘÍPADOVÉ STUDIE

Vzhledem k tomu, že společnost RailCo již navrhla dříve vymodelované aplikační služby, a že následné sekvenční diagramy neodhalily potřebu pro jakékoli další služby, nejsou zde žádné další procesní požadavky zahrnující jiné služby. Nicméně díky tomu, že rozhraní služby Invoice Processing (zpracování faktur) bylo vylepšeno tak, aby podporovalo budoucí rozšíření, je nyní schopné jako vstupní parametr přijímat také předem transformované dokumenty faktur.

V naší původní logice pracovního toku jsme v tomto procesu ustavili určitý krok, který dává službu Transform (převod) do pozice, kdy se stará jak o validaci, tak také o transformaci. Je tedy nutné upravit logiku pracovního toku tak, aby byla spolupráce se službou Transform (převod) volitelná pro případ, kdy operace Submit (předlož) služby Invoice Processing (zpracování faktur) obdrží na svém vstupu dokument faktury (protože ten je již transformován, a protože k validaci dokumentu dojde při jeho přijetí). Tato změna se jiných služeb nedotkne, protože vyžaduje pouze přidání nového podmíněného kroku zpracování do aplikační logiky zapouzdřené službou Invoice Processing (zpracování faktur).

PŘÍPADOVÉ STUDIE (PROCESS RESULTS)

Následuje finální verze definice služby Invoice Processing (zpracování faktur) zahrnující změny názvů elementů a všech předchozích revizí.

```
<definitions name="InvoiceProcessing"
   targetNamespace="http://www.xmltc.com/railco/transform/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:inv="http://www.xmltc.com/railco/invoice/schema/"
   xmlns:invs="http://www.xmltc.com/railco/invoiceservice/schema/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.xmltc.com/railco/transform/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <types>
      <xsd:schema targetNamespace=
         "http://www.xmltc.com/railco/invoiceservice/schema/">
         <xsd:import namespace=
            "http://www.xmltc.com/railco/invoice/schema/"
            schemaLocation="Invoice.xsd"/>
         <xsd:element name="SubmitInvoiceType">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="ContextID" type="xsd:integer"/>
                  <xsd:element name="InvoiceLocation" type="xsd:string"/>
```

```
                    <xsd:element name="InvoiceDocument"
                       type="inv:InvoiceType"/>
                 </xsd:sequence>
              </xsd:complexType>
           </xsd:element>
        </xsd:schema>
    </types>
    <message name="receiveSubmitMessage">
       <part name="RequestParameter"
          element="invs:SubmitInvoiceType"/>
    </message>
    <portType name="InvoiceProcessingInterface">
       <documentation>
          Zahájí proces Předložení faktur.
          Vyžaduje budoucí umístění dokumentu faktury
          nebo samotný dokument.
       </documentation>
       <operation name="Submit">
          <input message="tns:receiveSubmitMessage"/>
       </operation>
    </portType>
    ...
</definitions>
```

Výpis 15.21. Hotová abstraktní definice služby

Ti z vás, kterým je jazyk XML Schema Definition Language dobře známý, si možná povšimnou příležitosti k zavedení konstruktu `choice` kolem elementů `InvoiceLocation` (umístění faktury) a `InvoiceDocument` (dokument faktury). Pokročilé prvky jazyka XSD, jako jsou elementy `choice` a `union`, mají při použití v rámci WSDL definic omezenou podporu výrobců. Před jejich použitím si raději vše řádně ověřte.

Poznámka

Výsledkem tohoto procesu je pouze abstraktní definice. Kompletní WSDL dokument včetně konkrétních detailů definice a importovaného schématu lze stáhnout na adrese *www.serviceoriented.ws*.

SHRNUTÍ HLAVNÍCH POZNATKŮ

- Primárním činitelem při návrhu úlohově zaměřených řídicích služeb je přesná reprezentace logiky řídicího procesu, kterou mají tyto služby provádět.
- Znovupoužitelnost a spekulativní rozšíření návrhu jsou až druhotnými záležitostmi.

Pravidla pro návrh služeb

Začlenění servisně orientovaných principů návrhu do formálních standardů je pro úspěšné nasazení SOA v rámci dané organizace naprosto kritické. Tato část nabízí sadu pravidel, která lze použít jako výchozí bod, od něhož můžete odvodit své vlastní standardy.

Aplikujte standardy pro pojmenování

Označování služeb lze přirovnat k označování infrastruktury IT. Je proto zcela nezbytné, aby rozhraní služeb bylo v maximální možné míře konzistentně samopopisné.

Standardy pro pojmenování je tedy nutné definovat, a poté je aplikovat na:

- jména koncových bodů služeb,
- jména operací služeb,
- na hodnoty zpráv.

Existující konvence pro pojmenování se v každé organizaci liší. Některé využívají objektově orientované standardy pro pojmenování, kdy se objektům přiřazují podstatná jména a metody se označují slovesy. Jiné jednoduše používají slovesa jak pro komponenty, tak i pro jejich metody. I když by to bylo velice užitečné, žádný dokonalý standard vhodný pro všechny organizace neexistuje. Podstatné je, že jakékoliv zvolené standardy musejí být konzistentně implementovány napříč všemi prostředími se servisně orientovaným řešením.

Následující seznam uvádí některé podněty:

- Názvy kandidátů služeb s vysokým potenciálem pro znovupoužití napříč aplikacemi by měly být vždy zbaveny jakýchkoli charakteristik, jež poukazují na obchodní procesy, pro které byly původně sestaveny. Například místo pojmenování operace `GetTimesheetSubmissionID` raději použijte kratší `GetTimesheetID` nebo jen `GetID`.
- Aplikační služby je nutné pojmenovávat podle procesního kontextu, pod nímž jsou jejich operace seskupeny. Lze použít jak sloveso s podstatným jménem tak jen podstatné jméno. Zjednodušenými příklady vhodných názvů aplikačních služeb jsou `CustomerDataAccess`, `SalesReporting` a `GetStatistics`.
- Operace aplikačních služeb potřebují zřetelně sdělovat povahu své individuální funkcionality. Jako příklady vhodných jmen operací aplikačních služeb lze uvést `GetReport`, `ConvertCurrency` a `VerifyData`.
- Entitně zaměřené řídicí služby musejí představovat entitní modely, od nichž byli odvozeni jim odpovídající kandidáti. Proto musejí použité konvence pro pojmenování odrážet ty, jež byly stanovené v původním entitním modelu organizace. Tyto typy

služeb pro svá jména obvykle používají pouze podstatná jména. Příkladem vhodného jména entitně zaměřené řídicí služby jsou Invoice, Customer a Employee.

- Operace služeb pro entitně zaměřené řídicí služby by měly být založeny na slovesech a neměly by opakovat název příslušné entity. Kupříkladu entitně zaměřená služba nazvaná Invoice by neměla mít operace s názvem AddInvoice.

Aplikujte vhodnou úroveň členění rozhraní

Jak dokazují příklady případové studie v této kapitole, členění, ve kterém je daná služba navrhována, se může lišit. Sklon ke tvorbě rozhraní pro webové služby, které jsou hrubější než ta, jež jsou tradičně navrhována pro komponenty na bázi RPC, propagují výrobci jako prostředek pro překonání určitých problémů s výkonem spojených se zpracováním XML dat.

Výkon je samozřejmě pro úspěšný vývoj servisně orientovaných řešení kritický. Nicméně je třeba vzít v potaz také další činitele. Čím je totiž členění rozhraní hrubší, tím méně znovupoužitelnosti je schopno nabídnout. Je-li do jediné operace zabaleno více funkcí, může to být nevhodné pro žadatele, kteří potřebují pouze jednu z těchto funkcí. Kromě toho, některá hrubě členěná rozhraní si ve skutečnosti mohou vynucovat redundantní zpracování či výměnu dat tím, že nutí žadatele odesílat data nepodstatná pro konkrétní činnost.

Členění rozhraní služby je klíčovým strategickým rozhodnutím, které si během fáze servisně orientovaného návrhu zaslouží patřičnou pozornost. Následuje několik pravidel pro řešení těchto problémů:

- Snažte se plně porozumět faktorům omezujícím výkon cílového vývojového prostředí a prozkoumejte alternativní podpůrné technologie (jako jsou rozšíření pro binární kódování vyvinuté organizací W3C), je-li to vyžadováno.
- Prozkoumejte možnosti pro poskytnutí alternativních (hrubě a méně hrubě členěných) WSDL definic pro tytéž webové služby. Nebo prozkoumejte možnosti pro podporu redundantních hrubě a méně hrubě členěných operací v rámci téže WSDL definice. Tyto přístupy sice denormalizují kontrakty služeb, mohou však vyřešit problémy s výkonem a vyjít vstříc širokému spektru žadatelů.
- Přiřaďte hrubě členěná rozhraní službám označeným jako koncové body řešení a umožněte jemněji členěná rozhraní pro služby omezené na předem definované hranice. To je samozřejmě poněkud v protikladu k principům servisní orientace a charakteristikám SOA, jež podporují znovupoužitelnost a interoperabilitu služeb. Interoperabilita je podporovaná hrubě členěnými službami a o znovupoužitelnost se starají spíše jemněji členěné služby. Mohl by se také standardizovat model kompozice, který vyžaduje, aby hrubě členěné služby fungovaly jako řadiče a koncové body pro jemněji členěné služby.

Bez ohledu na zvolený přístup dbejte na to, aby byl konzistentní a předvídatelný, aby tak SOA mohla splnit požadavky kladené na výkon, a přitom zůstat standardizovaná.

PŘÍPADOVÉ STUDIE

Společnost TLS volí takový přístup pro členění rozhraní, při němž by služby vytvořené pro žadatele mimo společnost TLS poskytovaly konzistentní hrubě členěná rozhraní. Operace těchto služeb by přijímala všechna data nezbytná pro zpracování konkrétní činnosti. Dále by

cyklus výměny zpráv mezi externím žadatelem a službou byl vyžadován pouze tehdy, pokud by to bylo absolutně nezbytné nebo pokud by to vyžadovaly interní zásady společnosti.

Služby používané v rámci společnosti TLS by mohly poskytovat méně hrubě členěných operací pro usnadnění znovupoužitelnosti a širší paletu potenciálních (interních) žadatelů, tedy alespoň do té míry, dokud by režie zpracování vynucovaná méně hrubě členěnými operacemi byla přijatelná.

Při návrhu operací služeb dbejte na jejich rozšiřitelnost

Bez ohledu na to, jak dobře jsou služby při prvním nasazení navrženy, nikdy nebudou plně připraveny na to, co přinese budoucnost. Některé typy obchodních procesů změní výsledek kvůli nutnosti rozšíření rozsahu entit. Výsledkem pak je, že může být nutné rozšířit odpovídající řídicí služby. Zatímco charakteristiky služby, jako je znovupoužitelnost a komponovatelnost, jsou během rozdělování logiky jako části procesu modelování služeb promýšlené, rozšiřitelnost spadá spíše do fyzického návrhu, a proto musí být během návrhu brána v potaz.

V závislosti na povaze změny lze rozšiřitelnosti někdy dosáhnout bez rozbití stávajícího rozhraní služby. Je důležité navrhovat operace a zprávy tak, aby byly maximálně agnostické vzhledem k aktivitě. To podporuje zpracování budoucích nespecifických hodnot a funkcí, které stále souvisejí s celkovým účelem operace či zprávy. Kromě toho je dobrým zvykem reagovat na nové procesní požadavky nejdříve prošetřením možnosti kompozice dalších služeb, které jsou již k dispozici (včetně služeb, jež lze zakoupit či pronajmout). Tímto způsobem lze splnit požadavky bez zásahu do rozhraní služby.

Všimněte si, že rozšíření stávajícího rozhraní služby bude mít dopad na odpovídající XSD schéma. Tato rozšíření si lze ulehčit doplněním nových schémat určených speciálně pro ně. Ovšem před postupem tímto směrem se ujistěte, že máte pevně ustanovené standardy řízení verzí.

PŘÍPADOVÉ STUDIE

Vzhledem k velikosti společnosti TLS není neobvyklé, že jsou zaměstnanci přeřazováni, nebo že hledají vertikální či laterální změny pracovní pozice. Laterální scénář se díky heslu „prosazuj zevnitř" podněcovanému řadou vedoucích pracovníků stává stále běžnější.

Když zaměstnanec změní pozici nebo stupeň, očekává se od něj, že aktualizuje svůj profil pomocí formuláře na místním intranetu. Protože je však tento krok volitelný, je velmi často opomíjen. To, jak jinak, vede ke zvyšujícímu se počtu zastaralých profilů. Pro zamezení tomuto trendu upravila společnost TLS proces Předložení pracovních výkazů tak, aby obsahoval krok pro ověření profilu zaměstnance. Jakmile bude implementován, bude před přijetím pracovního výkazu verifikovat informace profilu. Pracovní výkazy předložené zaměstnanci s neplatnými profily budou jednoduše zamítnuty.

K implementaci tohoto nového požadavku není nutné měnit rozhraní služby Timesheet (pracovní výkaz). Místo toho se vylepší logika služby, do níž se začlení samostatná aplikační služba, která provádí ověření profilu.

Identifikujte známé i potenciální žadatele služby

Služby jsou téměř vždy budované jako součást dodávky specifického řešení automatizace. Bývají tedy navržené pro řešení podnikových požadavků tak, jak jim náleží v dané aplikaci. Omezení návrhu určité služby na splnění bezprostředních požadavků může bránit jejímu potenciálu jako znovupoužitelné, adaptivní a interoperabilní jednotce procesní logiky.

Je tedy žádoucí, aby jakýkoliv stávající proces návrhu služby zahrnoval spekulativní analýzu toho, jakým způsobem může být tato služba využita vně jejích počátečních aplikačních hranic. Jinými slovy, může být užitečné a praktické identifikovat jakékoliv potenciální budoucí žadatele služby a začlenit jejich očekávané požadavky do současného návrhu služby.

To může vést k dalším funkčním požadavkům, které mohou nebo nemusejí být žádoucí, v závislosti na aktuálním rozsahu projektu, rozpočtu a dalších s tím souvisejících omezeních. Co je však důležitější, může to vést k návrhovým vytříbenostem, které vůbec nemusejí mít zásadní dopad na aktuální projekt. Může být například možné ve fázi návrhu upravit členění rozhraní služeb bez výrazného dopadu na celkový projekt.

Zvažte používání modulárních WSDL dokumentů

Popisy WSDL služeb lze dynamicky sestavovat za běhu pomocí příkazů `import`, které připojí samostatné soubory obsahující části definice služby. To vám umožní definovat modely pro typy, operace a vazby, které můžete sdílet mezi WSDL dokumenty.

Umožní vám to také využívat jakékoli stávající moduly XSD schémat, které jste již navrhli. Řada organizací odděluje schémata do členěných modulů, jež představují jednotlivé složené typy. To utváří centralizované úložiště schémat, která lze sestavovat do vlastních hlavních definic schémat. Díky možnosti importovat moduly XSD schémat do konstruktu `types` WSDL definice, mohou nyní vaše vlastní WSDL dokumenty používat tytéž moduly se schématy.

Poznámka

Ostatně specifikace WS-I Basic Profile vyžaduje, aby se při návrhu modulárních WSDL definic používal příkaz `import` pro import dalších WSDL definic či XSD schémat.

PŘÍPADOVÉ STUDIE

Společnost TLS zvažuje importování konstruktu `bindings`, aby mohl být opětovně použit a snad i určován dynamicky. Nicméně, později bylo rozhodnuto ponechat konstrukt `bindings` jako součást WSDL dokumentu. Níže je uveden příkaz `import` používaný pro provedení tohoto testu:

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

Výpis 15.22. Element import používaný pro importování konstruktu bindings sídlícího v samostatném souboru

Obory názvů (jmenné prostory, namespaces) používejte obezřetně

WSDL definice sestává z kolekce elementů s různým původem. Proto každá definice často zahrnuje několik různých oborů názvů. Následuje seznam obvyklých oborů názvů používaných k reprezentaci specifikačně založených elementů:

- `http://schemas.xmlsoap.org/wsdl/`
- `http://schemas.xmlsoap.org/wsdl/soap/`
- `http://www.w3.org/2001/XMLSchema/`
- `http://schemas.xmlsoap.org/wsdl/http/`
- `http://schemas.xmlsoap.org/wsdl/mime/`
- `http://schemas.xmlsoap.org/soap/envelope/`

Při sestavování WSDL dokumentu z modulů přicházejí ke slovu dodatečné obory názvů, zejména pak při importu definicí XSD schémat. Kromě toho můžete při definování vašich vlastních elementů stanovit více oborů názvů pro reprezentaci aplikačně specifických částí WSDL dokumentů. U větších WSDL dokumentů není neobvyklé, aby obsahovaly až deset odlišných oborů názvů a kvalifikátorů, které je mají doprovázet. Proto je nanejvýš vhodné, abyste si používání oborů názvů uvnitř i mezi WSDL dokumenty pečlivě zorganizovali.

Specifikace WS-I Basic Profile vyžaduje použití atributu `targetNamespace` pro přiřazení oboru názvů WSDL dokumentu jako celku. Je-li ve WSDL definici vloženo XSD schéma, pak tato specifikace požaduje, aby měla také definován atribut `targetNamespace` (který může obsahovat tutéž hodnotu, jako jeho protějšek určený pro celý WSDL dokument).

PŘÍPADOVÉ STUDIE

Některé z běžných oborů názvů identifikovaných dříve nejsou vyžadovány službou Employee (zaměstnanec) společnosti TLS, a proto jsou ze seznamu atributů elementu `definitions` vypuštěny. Je přidán atribut `targetNamespace` společně se dvěma obory názvů spojenými se dvěma importovanými schématy.

```
<definitions name="Employee"
   targetNamespace="http://www.xmltc.com/tls/employee/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:act="http://www.xmltc.com/tls/employee/schema/accounting/"
   xmlns:hr="http://www.xmltc.com/tls/employee/schema/hr/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.xmltc.com/tls/employee/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   ...
</definitions>
```

Výpis 15.23. Deklarace oborů názvů v rámci elementu definitions v souboru Employee.wsdl společnosti TLS

Používejte SOAP dokumenty a atributům přiřazujte hodnotu „literal"

Existují dva specifické atributy, které ustavují formát obsahu SOAP zprávy a systém datových typů používaný pro reprezentaci obsahových dat. Jedná se o atribut `style` požívaný elementem `soap:binding` a atribut use přiřazený elementu `soap:body`. Oba tyto elementy sídlí uvnitř WSDL konstruktu `binding`.

Nastavení těchto atributů je důležité, neboť souvisí se způsobem, jakým je obsah SOAP zprávy strukturován a reprezentován.

- Atributu `style` lze přiřadit hodnotu „document" nebo „rpc". První z nich podporuje vkládání celých XML dokumentů do SOAP těla, zatímco druhá je navržena spíše pro tradiční RPC komunikaci, a proto podporuje data parametrového typu.
- Atribut `use` lze nastavit na hodnotu „literal" nebo „encoded". Protokol SOAP původně nabízel vlastní systém typů používaný pro reprezentaci obsahu těla zprávy. Později byla začleněna podpora pro datové typy jazyka XSD. Hodnota tohoto atributu signalizuje, jaký systém typů má vaše zpráva používat. Nastavení „literal" říká, že se mají aplikovat datové typy jazyka XSD.

Protokol SOAP podporuje následující čtyři možné kombinace těchto dvou atributů:

- `style:RPC` + `use:encoded`
- `style:RPC` + `use:literal`
- `style:document` + `use:encoded`
- `style:document` + `use:literal`

SOA preferuje kombinace `style:document` + `use:literal`, jež podporuje model předávání zpráv v dokumentovém stylu, který je významný pro realizaci prvků mnoha klíčových specifikací WS-*. Kromě toho specifikace WS-I Basic Profile vyžaduje, aby byl atribut `use` nastaven vždy na hodnotu „literal".

PŘÍPADOVÉ STUDIE

Při sestavování konkrétní části definice rozhraní služby Employee (zaměstnanec) se architekti společnosti TLS rozhodli použít kombinaci `style:document` + `use:literal`:

```
<binding name="EmployeeBinding"
   type="tns:EmployeeInterface">
   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetWeeklyHoursLimit">
      <soap:operation soapAction="http://www.xmltc.com/soapaction"/>
      <input>
         <soap:body use="literal"/>
      </input>
```

```
      <output>
         <soap:body use="literal"/>
      </output>
   </operation>
   <operation name="UpdateHistory">
      <soap:operation soapAction="http://www.xmltc.com/soapaction"/>
      <input>
         <soap:body use="literal"/>
      </input>
      <output>
         <soap:body use="literal"/>
      </output>
   </operation>
</binding>
```

Výpis 15.24. Konstrukt binding z dokumentu Employee.wsdl společnosti TLS

Používejte profily WS-I i bez požadavku na dodržení specifikace WS-I

Není-li vyhovění specifikaci WS-I na vašem seznamu bezprostředních požadavků, je i přesto doporučováno, abyste zvážili používání mnoha standardů a osvědčených postupů nabízených dokumentem Basic Profile. Jsou spolehlivé, dobře probádané a osvědčené a mohou vám ušetřit obrovskou spoustu času a úsilí při vývoji vašich vlastních standardů.

Poznámka

Dalším výsledkem specifikace WS-I, které se v této knize nevěnujeme, je profil Basic Security Profile, jenž řídí a standardizuje používání bezpečnostních specifikací pro účely interoperability. Více informací naleznete na stránce *www.ws-i.org*.

Dokumentujte služby pomocí metadat

Jak dosvědčuje diskuse o specifikacích WS-Policy (zásady webových služeb) a WS-MetadataExchange (výměna metadat v rámci webových služeb) v kapitole 7, platforma WS-* ve stále větší míře zdůrazňuje kvalitu a hloubku popisů služeb. Pro mnoho architektur SOA nebude výjimečné existovat bez výhody technického prostředí schopného podporovat obsah popisu služby za hranici poskytovanou WSDL definicemi.

Zásady představují důležitý doplněk ve formě metadat k WSDL definicím. Zásady mohou například vyjadřovat určité bezpečnostní požadavky, procesní priority a charakteristiky chování poskytovatele dané služby. Žadatelům služby to umožňuje lépe ohodnotit poskytovatele služby a k tomu se jim dostává příležitost plně se na vzájemnou spolupráci připravit.

Zásady jsou formálně implementovány pomocí sady specifikací WS-* popsaných v kapitole 7. Bez ohledu na to, zda se tyto specifikace v dané organizaci skutečně používají, by i tak měly být informace o zásadách zdokumentované jako součást návrhu služby. Jednak to totiž vývojářům budujícím žadatele služby pro daného poskytovatele služby nabízí velké množství užitečných informací, a jednak to usnadňuje plynulý přechod v momentě, kdy se zásady stanou běžnou součástí servisně orientovaných architektur.

Všechny unikátní vlastnosti služby by měly být zdokumentovány pro snazší vysvětlení požadavků, charakteristik či omezení služby ostatním, kteří mohou chtít danou službu využívat. Tuto informaci lze přidat do WSDL definice pomocí elementu `documentation`. Mohla by dokonce být obsažena v rámci dokumentu s metadaty, který je zveřejněn samostatně, a jenž je snadno přístupný. To vše vede k podpoře zjistitelnosti a znovupoužitelnosti služeb. Kdykoli je to možné, může být tato dokumentace také fyzicky připojena k elektronickým modelovacím diagramům.

Poznámka

Pokud navrhujete služby vyhovující specifikaci WS-I, můžete zvýšit kvalitu popisu jejich metadat připojením sdělení o splnění specifikace WS-I. Ta pak oznamují dodržení určitých profilů WS-I. Více informací najdete na adrese *www.ws-i.org*.

SHRNUTÍ HLAVNÍCH POZNATKŮ

- Při návrhu webových služeb je nutné docílit vyváženosti mezi splněním požadavků vyjádřených v modelech kandidátů služeb a vynucovaných činiteli reálného světa.
- Pro budování kvalitních služeb, které se drží stávajících standardů pro návrh rozhraní, je doporučováno zahájit jejich návrh tvorbou WSDL definice.