VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

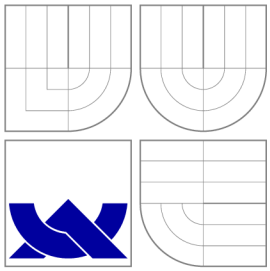
WEBOVÉ SLUŽBY JAKO REALIZACE ARCHITEKTURY ORIENTOVANÉ NA SLUŽBY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

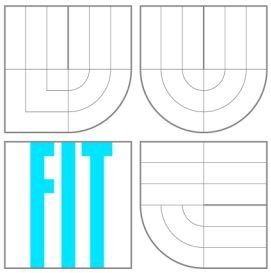
AUTOR PRÁCE
AUTHOR

Bc. PAVEL JIRÁČEK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÉ SLUŽBY JAKO REALIZACE ARCHITEKTURY ORIENTOVANÉ NA SLUŽBY

REALISATION OF SERVICE-ORIENTED ARCHITECTURE: WEB SERVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PAVEL JIRÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR WEISS

BRNO 2009

Abstrakt

Tématem diplomové práce je architektura orientovaná na služby (SOA). Zprostředkovává celkový přehled webových služeb jako realizace SOA. Práce zmiňuje, co je připraveno a co schází k naplnění základních principů SOA. Analyzuje nástroje platformy Java EE pro webové služby. Představuje technologii Java Management Extensions (JMX) pro řešení jedné z oblastí webových služeb - SOA governance. Analyzuje možnosti použití JMX pro řízení a monitorování webových služeb. Za účelem testování implementuje základní aplikaci pro řízení a monitorvání webových služeb na serveru GlassFish.

Abstract

The main subject of this master thesis is the Service-Oriented Architecture (SOA). It shows a complete overview of Web Services in terms of realisation of a SOA. The thesis mentions what is already done and what else remains to fulfill the basic SOA principles. Then, it reviews Java EE platform tools used to deal with the Web Services. Next, it introduces the Java Management Extensions (JMX) technology as a solution of the web service's area - SOA governance. Moreover, it examinates the possible usage of JMX for the web service's management and monitoring issues. Finally, it implements an exemplary web services management and monitoring application on GlassFish server for testing purposes.

Klíčová slova

architektura orientovaná na služby, webové služby, WSDL, SOAP, UDDI, Java EE, management, monitorování, SOA governance, GlassFish, JMX, AMX, mbeana

Keywords

Service-Oriented Architecture, web services, WSDL, SOAP, UDDI, Java EE, management, monitoring, SOA governance, GlassFish, JMX, AMX, mbean

Citace

Pavel Jiráček: Webové služby jako realizace architektury orientované na služby, diplomová práce, Brno, FIT VUT v Brně, 2009

Webové služby jako realizace architektury orientované na služby

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Weisse

.....
Pavel Jiráček
26.5.2009

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Petru Weissovi za trpělivost při konzultacích a pomoc při vypracovávání diplomové práce.

Obsah

Kapitola 1

Úvod

Obor informačních technologií prošel v posledních letech výrazným rozvojem. Za tuto dobu se s vyspělostí implementací jednotlivých systémů přímo úměrně zvyšovaly nároky kladené na jejich funkcionalitu a celkově na technologie použité k vývoji i provozu. Informační systémy pronikly do všech oblastí běžného života. Významný posun zaznamenala hlavně sféra podnikových řešení. Zde se postupně navyšuje podíl zpracovávaných dat a operací vykonávaných systémem. Pro dosažení vyšších zisků se stále více investuje do automatizace běžných úkonů. Neexistuje však nikdo, kdo by řídil a snažil se sjednotit použité postupy a techniky návrhu a vývoje. Vedlejším efektem tak vzniká množství komplexních systémů, jejichž nešvarem je značná různorodost. Otázka jejich pozdějšího propojení a kooperace bývá často neřešitelná.

Určitou cestou, jak řešit integraci více nesourodých systémů, je použití architektury orientované na služby (SOA - z angl. Service Oriented Architecture). Díky SOA lze zjednodušit vývoj, údržbu i následnou modernizaci systémů a dosáhnout tak celkového snížení nákladů. SOA definuje strukturu podnikových aplikací složených z více komunikujících modulů nezávislých na vzájemné implementaci. Moduly, fungující jako samostatné služby, tak neztrácejí výhody získané použitím specifických technologií.

Tato diplomová práce se zabývá tématem webových služeb jako jedné z formy realizace architektury orientované na služby. Zprostředkovává obecný přehled o webových službách a hodnotí je z hlediska naplnění hlavních myšlenek SOA. Zmiňuje, co je realizováno a co schází k úplné realizaci architektury. Následně se zabývá platformou Java EE a jejími prostředky v oblasti implementace webových služeb. Z hlavních nevyřešených otázek si vybírá oblast SOA governance a její součást týkající se managementu a monitorování. Představuje technologii Java Management Extensions (JMX) z pohledu obecných aplikací Javy. Zkouší otestovat možnosti jejího využití v rámci managementu a monitorování webových služeb na serveru GlassFish. V rámci provedení testu popisuje, jakým způsobem provádět základní kroky pro úpravu služeb i vytvoření elementární aplikace pro jejich řízení a sledování. Snaží se analyzovat hlavní klady i zápory technologie ve spojení s webovými službami a zmínit případné nedostatky v její implementaci. Diplomová práce nemá za úkol navrhnout komplexní aplikaci pro správu služeb ani popsat všechny možnosti aplikace JMX.

Text práce je rozdělen do sedmi hlavních kapitol. Kapitola č.2 popisuje obecné principy architektury orientované na služby. Ukazuje základní rozložení prvků v systémech SOA a hodnotí její klady a zápory. Zahrnuje také úvod do problematiky SOA governance.

Následující kapitola č.3 je věnována webovým službám. Popisuje služby v souvislosti se SOA. Uvádí základní standardy tvořící jádro webových služeb. Pokračuje popisem rozšíření, jež přibližují webové služby kompletní realizaci SOA. Otázky SOA, které je třeba v rámci

webových služeb dořešit.

Kapitola číslo 4 definuje možnosti platformy Java Enterprise Edition v oblasti implementace webových služeb. Popisuje způsob práce se službami s využitím základních dvou aplikačních rozhraní. Nakonec zmiňuje spojení Javy a SOA Governance.

Obecný úvod do technologie Java Management Extensions je situován do kapitoly s číslem 5. Zde je popsán základní princip JMX. Součástí následujícího přehledu architektury JMX systémů je popis všech jejích vrstev. Konec kapitoly je věnován Java EE Management Modelu jako prostředku k unifikaci J2EE serverů.

Po vysvětlení základu JMX se v kapitole č.6 dostáváme k jejímu použití. Hlavním tématem je zde prvek managementu zvaný mbeana. Nastiňujeme její strukturu, vlastnosti a účel v rámci systému. Dále uvádíme cesty k propojení řídící aplikace se serverem. Poslední část kapitoly ukazuje způsoby manipulace s mbeanami.

Předposlední kapitola č.7 je orientovaná na využití technologie JMX na serveru GlassFish. Zmiňuje specifika serveru v oblasti management aplikací spojená s jeho aplikačním rozhraním Application Server Management Extensions (AMX). Popisuje informace, které GlassFish poskytuje o webových službách formou generických mbean. V závěru se zabývá tématikou monitorování.

Poslední kapitola s číslem 8 je věnována testování spojení webových služeb s řídícími systémy JMX formou základní aplikace. Snaží se shrnutí informací získaných při implementaci aplikace a zmínění problémů s implementací spojených.

Kapitola 2

Architektura orientovaná na služby

2.1 Hlavní idea architektury

Architektura orientovaná na služby představuje způsob návrhu aplikací, definuje infrastrukturu, nad níž je možné jednoduše a účinně vytvářet nové aplikace. Ukazuje způsob, jak využít výhod rozličných softwarových architektur a jejich specifického modelování, řízení nebo nástrojů pro vývoj. Představuje odpovídající úroveň abstrakce, potřebnou pro svázání potřeb podniku s jeho technickým zázemím.

Celá aplikace se skládá ze vzájemně komunikujících výpočetních prvků, nazývaných služby. Tyto služby mohou mezi sebou komunikovat přes jednotná veřejná rozhraní pomocí zasílaných zpráv. Na základě těchto zpráv se vytváří tzv. popis službu, který je tedy oddělen od detailů jejího aplikačního prostředí. Kód implementace je od konzumenta služby oddělen komunikačním rozhraním. Architektura dovoluje vzájemně kombinovat kód různých platforem umístěný v odlišných lokalitách, tím je využito specifických výhod různých aplikačních prostředí.

Použití architektury orientované na služby vede k silné rozdistribuovanosti výpočetních prostředků a tím k rovnoměrnému rozložení zátěže. Každý prvek konzumující službu se může stát jejím poskytovatelem. Aplikační logika je rozmístěna mezi všechny prvky zapojené do komunikace a narozdíl od architektury klient-server neobsahuje žádný centrální server, kam by byla soustředěna veškerá výpočetní síla.

Specifické principy užité při návrhu a implementaci služeb v rámci architektury orientované na služby jsou podle [13] a také [6] následující:

- abstrakce služeb - implementační logika je schována před vnějším světem. Pro jedno konkrétní rozhraní služby může existovat mnoho implementací různých aplikačních prostředí.
- zapouzdření služeb - služby zapouzdřují svou funkcionalitu na různých úrovních.
- volné vázání služeb - vztahy mezi službami jsou udržovány na minimální nutné úrovni. V zásadě se informace zveřejněné o službě týkají jen její existence a dat poskytovaných veřejným rozhraním. Tato vlastnost podstatně zjednodušuje modernizaci či úplnou výměnu služeb, kdy je jedna implementace zaměněna za druhou při zachování komunikačního rozhraní.
- kontrakt služeb - služby dodržují podmínky vzájemné komunikace dané popisem.

- znovupoužitelnost služeb - ne záměrným, ale vítaným vedlejším efektem je možnost opětovného použití služeb. Při správném postupu návrhu a implementace se tak dá využít dříve vytvořených zdrojů. což má za následek úsporu nákladů na vývoj.
- skládatelnost služeb - služby se dají vzájemně kombinovat. Vznikají tak služby nové, které často provádějí komplexnější, náročnější úkoly.
- autonomie služeb - služby jsou samostatné jednotky, které se dokáží nezávisle rozhodovat a samy řídit.
- optimalizace služeb - používá se preference kvalitnějších služeb nad ostatními.
- objevitelnost služeb - služby jsou navrhovány jako sebepopisné. Vnější elementy starající se o vyhledávání služeb mají šanci uveřejněnou službu nalézt a podle popisu využít.

2.2 Struktura vrstev architektury

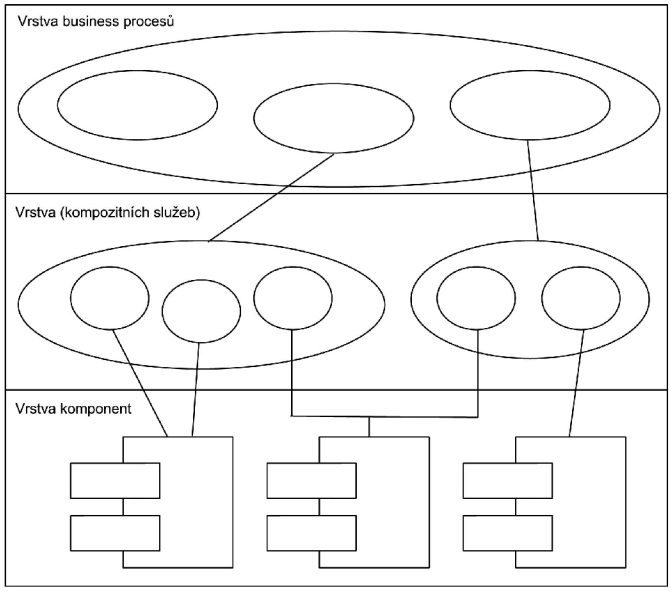
Při použití architektury orientované na služby lze na aplikaci nahlížet jako na strukturu vrstev s různou úrovní abstrakce. Názorně ji ilustruje obrázek 2.1. Na nejvyšší úrovni jsou podnikové procesy. Ty tvoří klíčové prvky při tvorbě softwaru. Vše je podřízeno modelovaným podnikovým procesům. Každý proces představuje určitou aktivitu uvnitř podniku, která je významná nejčastěji z důvodu produkce nějakého zisku. Samotné podnikové procesy jsou na nižší úrovni abstrakce mapovány na jednu či více sdružených služeb. Služba je abstrakcí nějaké funkční jednotky s různou výpočetní silou. Elementární služby provádějí velmi jednoduché úkony. Složitější záležitosti se řeší vzájemnou kombinací jednodušších služeb. Vzniká tak ucelená hierarchická struktura. Implementace služeb je zajištěna pomocí komponent ležících na nejnižší vrstvě abstrakce. Komponenty jsou poté navrženy pomocí procedurálního, objektově-orientovaného či jiného paradigma.

2.3 Komunikační model

Základním kamenem pro použití služeb je jejich vzájemná spolupráce. Bez vzájemné kooperace služeb by celá architektura postrádala smysl. Zde je nutné uvést zavedení rolí *spotřebitele* a *poskytovatele* služby pro obě komunikující strany. Poskytovatel je ten, kdo nabízí k použití danou službu, přičemž spotřebitel je druhá strana, která službu využívá. Role spotřebitele i poskytovatele může zastávat jakákoliv služba. Komunikace se účastní zpočátku i třetí strana a tou je registr služeb.

Celý proces komunikace všech tří účastníků probíhá následujícím způsobem:

1. Poskytovatel naimplementuje službu určenou ke zveřejnění
2. Poskytovatel uveřejní informace o službě ve formě dokumentu tvořící jednotný popis služby. Popis obsahuje mimo jiné informace o umístění služby, nabízených metodách nebo datech které přijímá, zpracovává a navrací zpět. Uveřejnění probíhá zasláním dokumentu do společného registru služeb.
3. Spotřebitel, který se zajímá o konkrétní službu, kontaktuje registr s žádostí. Ta je zaslána ve formě zprávy obsahující detailní popis vlastností služby, kterou spotřebitel vyhledává.



Obrázek 2.1: Abstrakce struktury vrstev architektury SOA

4. Registr služeb je prohledán na základě přijaté žádosti. Výsledek je zaslán zpět spotřebiteli.

5. Spotřebitel podle doručeného popisu sestrojí přesnou zprávu a přímo kontaktuje službu na straně poskytovatele.

6. Další část komunikace probíhá přímo mezi spotřebitelem a poskytovanou službou a liší se podle použitého vzoru pro výměnu zpráv dané služby. Ve většině případů služba zpracuje příchozí zprávu a její obsah a vrací výsledek zpět spotřebiteli.

2.4 Vývoj aplikací orientovaných na služby

Při vývoji aplikací založených na službách se jen zřídkakdy postupuje „od čistého stolu“. Ve většině případů slouží jako základ stávající systém a ten je dekomponován. Podle [6] se při návrhu používá kombinace tří osvědčených postupů. Prvním z nich je návrh shora dolů. Analyzují se při něm podnikové procesy podniku, rozkládají na podprocesy, které jsou modelovány službami. Druhým přístupem je orientace zdola nahoru. Zde se právě využívá stávajícího systému k identifikaci služeb podle již vytvořených řešení. Modelování cílových služeb je přístupem posledním. Doplňuje analýzu shora dolů a zdola nahoru o dosud neimplementované služby, upravuje stávající služby nebo redukuje celkový počet služeb v navrhovaném řešení.

Návrh a celkový vývoj aplikací založených na architektuře orientované na služby využívá postupů známých z vývoje objektově-orientovaných, procedurálních nebo jinak zaměřených systémů. Ty jsou užitečné zejména při implementaci metod používaných službami. Služby samotné jsou ale definovány pomocí vzájemně vyměňovaných zpráv spíše než popisem metod, které používají. Navíc jsou použity mechanismy pro zajištění spolupráce služeb jakými jsou choreografie nebo orchestrace.

2.5 SOA Governance

Anglický pojem governance by se dal interpretovat jako vládnutí nebo řízení. Obecně se jedná o seskupení pravidel, zvyků a omezení, které slouží k řízení, správě a kontrole entity, nad kterou jsou definovány. SOA governance je pouze jednou částí v hierarchii řízení celého podniku. Jedná se pouze o část nadřazené IT Governance - řízení IT frakce podniku, která je opět jen součástí řízení celé společnosti - tzv. Corporate Governance.

Architektura orientovaná na služby určuje představu, jak má vypadat systém založený na službách. Definuje jeho správnou strukturu, popisuje vnitřní komunikaci a uvádí poskytované výhody celého řešení. SOA governance vznikla jako nástroj pro řízení služeb těchto systémů v průběhu jejich celého životního cyklu. Umožňuje kontrolu každé služby od návrhu a vývoje přes použití až k jejímu odstranění ze systému. Díky takovéto kontrole pak existuje jistota, že vytvořená služba splňuje veškeré funkční požadavky, neporušuje předem stanovená pravidla a vykonává svou funkci s nejvyšší možnou efektivitou.

Domény působnosti SOA governance by se daly rozdělit na:

- Dodržování standardů
- Identifikace a zařazení služby
- Poskytování služby
- Monitorování služby
- Schopnost spolupráce služeb

Ve mnohém se jedná o zásady spíše teoretického charakteru, na které by mělo být pohlíženo při návrhu a vývoji služeb. Dodržování těchto pravidel je pak otázkou ustanovených podnikových konvencí pro oblast vývoje aplikací architektury orientované na služby než procesem, který by se dal řídit programově.

K prosazování myšlenek SOA governance v oblasti monitorování a správy webových služeb se ale dají využít již existující prostředky. Kapitola 4 popisuje jeden z nich - aplikační konzoli pro správu zdrojů umístěných na aplikačním serveru GlassFish firmy Sun.

2.6 Výhody a nevýhody architektury

Při shrnutí veškerých základních principů, o které se architektura orientovaná na služby opírá, se zdá, že výraznější pozici mají klady před zápory. Při přechodu na tuto architekturu se jako v ostatních případech jedná o zisk. Toho je dosaženo několika cestami. Úspora nákladů je zapříčiněna využitím již existujících systémů. Snižuje se tím čas pro uvedení hotového produktu na trh. Znovupoužitelnost služeb zajišťuje redukci investic, při vývoji. Nižší prostředky jsou potřeba na provoz a údržbu systému. Rychlá a jednoduchá modifikace nebo vylepšení funkcionality připojením nových služeb poskytují požadovanou flexibilitu při reakcích na změny trhu.

Použitá technologie je více přizpůsobena lidem, kteří ji využívají. Zvyšuje se tak efektivita práce a úspěšnost projektu.

Mezi jasné nevýhody použití této technologie řadíme vyšší náklady při přechodu z jinak navržených systémů. Návrat investic je spíše dlouhodobý. Výhody se ukazují až po uplynutí určité doby, kdy je vytvořena dostatečná infrastruktura služeb, pokrývající základní potřeby daného podniku. Služby proto musí být navrhovány v širším kontextu použití, jinak se nedá

těžit z výhod jejich opětovného zapojení. Výhod architektury umí využít jen lidé, kteří vědí jak správně navrhnout daný systém.

Kapitola 3

Webové služby

3.1 Úvod

World Wide Web byl navržen a je užíván lidmi. Stal se velmi populárním díky své nenáročnosti na ovládání, široké dostupnosti a schopnosti poskytovat určitou interaktivitu. Celkově se jedná o zpřístupňování vzdálených informací a jejich transformaci do lidsky čitelné podoby. Nutnou součástí webu je vždy právě člověk, který vykonává onu interaktivitu - hledá v dokumentech, formuluje požadavky. Webové služby můžeme chápat jako obdobu webu přizpůsobenou strojům. Tedy bez přítomnosti člověka lze zajistit spolupráci dvou programů, které si navzájem vyměňují data, zpracovávají a dále je využívají.

Existuje několik různých definic webových služeb. Například mezinárodní organizace W3C (World Wide Web Consorcium), která zaštiťuje standardy týkající se Webu ve své specifikaci [14] definuje webovou službu jako *„Softwarový systém navržený pro podporu spolupráce mezi stroji přes síť. Má rozhraní popsané ve formátu čitelném strojem (typicky WSDL). Ostatní systémy spolupracují se službou způsobem předepsaným jejím popisem pomocí SOAP zpráv, typicky přenášených použitím HTTP protokolu s XML zápisem ve spolupráci s ostatními webovými standardy“*. Z trošku jiného pohledu se na webovou službu pohlíží v [5], kde definice zní: *„Webové služby jsou tedy webem pro stroje, umožňují interakci mezi programy běžícími kdekoliv v Internetu, aniž by musel do jejich konání zasahovat člověk.“* Každá definice se částečně liší od ostatních, všechny však shodně vyjadřují základní podstatu webových služeb. Jde o dosažení kooperace mezi dvěma a více systémy různých platforem použitím standardizovaných komunikačních rozhraní, jednotných protokolů a sjednocených formátů přenášených dat.

Webové služby jsou jednou z implementací architektury orientované na služby. Jak však upozorňuje [4], jakkoliv mohou webové služby implementovat tuto architekturu, neznamená to automaticky, že architektura orientovaná na služby je rovna webovým službám. I přes fakt, že webové služby jsou nejrozšířenější zprostředkování architektury SOA, existují i jiné systémy založené na stejném základě, které nemusí být nutně webové.

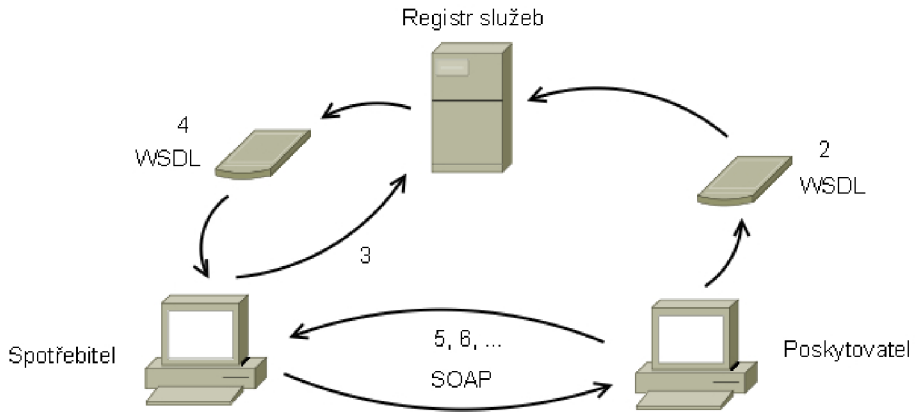
3.2 Fungování webových služeb

Webové služby se snaží kompletně naplnit myšlenku architektury orientované na služby. K využití jejích výhod se však dá dospět pouze cestou dodržování pevně daných standardů pro definici služeb, způsob a obsah jejich vzájemné komunikace. Mezi technologie webových služeb, které tvoří pilíře základní funkcionality patří:

- XML - eXtensible Markup Language
- WSDL - Web Services Description Language
- SOAP - Simple Object Access Protocol
- UDDI - Universal Description, Discovery and Integration

Popis jednotlivých technologií je situovaný do následujících kapitol.

Komunikace v případě webových služeb probíhá přesně podle vzoru uváděného SOA. Ilustruje ji obrázek 3.1. Webové služby podporují spojení poskytovatele služby s jejím spotřebitelem, doplněné o spolupráci s registrem nabízených služeb. Poskytovatel naimplementuje službu a její popis ve formě WSDL dokumentu uveřejní v registru UDDI. Tento dokument je pak zasílán spotřebiteli jako odpověď na žádost o nalezení požadované služby. Podle WSDL popisu dokáže spotřebitel zkonstruovat SOAP zprávu a započnout dialog s poskytovatelem. Veškerá komunikace, včetně žádostí do UDDI registru, probíhá ve formě SOAP zpráv založených na XML. Tento jazyk slouží jako základ jazyka WSDL a tudíž i jako prostředek pro popis rozhraní služby. XML tedy můžeme po právu prohlásit základním stavebním kamenem webových služeb, bez kterého by nebylo možné služby realizovat.



Obrázek 3.1: Komunikační model

Jazyky WSDL a XML spolu s komunikačním protokolem byly uznány za plnohodnotné standardy a zajišťují elementární funkce potřebné k provozu webových služeb. K základní realizaci jsou však postupně přidávány rozšíření, která umožní zavedení aplikací postavených na webových službách i tam, kde jsou vyžadovány vyšší nároky.

3.3 XML

3.3.1 Úvod

Jazyk XML (eXtensible Markup Language) byl organizací W3C uznán za standard. XML je, podobně jako HTML (HyperText Markup Language), jazykem značkovacím, sdílí shodného předka - jazyk SGML a pro webové služby plní podobnou úlohu, jako HTML pro web. Ovšem zatímco HTML bylo vytvořeno za účelem zobrazování dat, jazyk XML slouží k jejich definici. XML bylo vytvořeno jako jazyk sebepopisný a neobsahuje, narozdíl od HTML,

předem definovaný seznam povolených tagů. Právě slovo eXtensible dává uživateli možnost definovat si značky svého vlastního významu.

XML dokumenty jsou textové, nikoliv binární. Díky použití kódování Unicode dokáží reprezentovat téměř všechny lidmi používané znaky. Jednou z hlavních výhod, využívanou webovými službami, je nezávislost na platformě. To umožňuje přenášet data mezi službami, implementovanými v odlišných aplikačních prostředích.

XML reprezentuje data formou stromu. Vytváří tak hierarchii vzájemně zanořených elementů, jejich atributů i textového obsahu. Dává maximální volnost při vytváření struktury. Omezení je minimum a týkají se syntaxe i sémantiky jednotlivých prvků. V této oblasti se nejčastěji objevují následující dva pojmy:

- *Správná strukturovanost (z angl. „well-formedness“)* - Správně strukturovaný dokument splňuje všechny následující syntaktické podmínky:
  - dokument má kořenový element,
  - všechny elementy mají uzavírací tag,
  - názvy tagů jsou závislé na velikosti jejich písmen (z angl. „case-sensitive“),
  - elementy musejí být správně zanořované,
  - hodnoty atributů musejí být uzavřeny v uvozovkách.
- *Validita* - Validní XML dokument je každý takový, který splňuje veškeré sémantické podmínky definovány pro daný typ dokumentu. Prostředky, kterými se validita XML dokumentu kontroluje jsou tyto:
  - *Document Type Definition (DTD)* - Jazyk náležící do rodiny jazyků pocházejících z SGML. Nepodporuje modernější prvky XML dokumentů - např. jmenné prostory XML. I přes své omezené schopnosti je rozšířen díky své jednoduchosti použití.
  - *XML-Schema* - Novější a schopnější nástroj. XML Schema je používáno ve webových službách a jeho schopnosti jsou popsány níže.

Schopnosti jazyka XML jsou zvýrazněny jeho podporou mnoha rozšíření. Podle [15] můžeme jmenovat *XPath* pro navigaci uvnitř dokumentu a adresaci jeho elementů a atributů, také *XML Encryption* definující pravidla pro šifrování dokumentů. *XML Namespaces* a *XML Schema* jsou výraznými technologiemi v rámci webových služeb a jejich konkrétní popis následuje.

3.3.2 XML Namespaces

Jmenné prostory XML jako jedno z rozšíření slouží k upřesnění názvů uvnitř dokumentu. Vytváří tzv. plně kvalifikovaná jména. Tato jména se skládají z klasického jména doplněného o předponu zaručující příslušnost prvku do určitého jmenného prostoru. Dosahuje se tak jednoznačnosti názvů prvků při spojení více dokumentů a zabraňuje se kolizím shodných jmen s odlišnou sémantikou. Jmenné prostory XML si lze představit jako obdobu systému balíčků v Javě. Jmenné prostory XML jsou přítomny ve všech zmíněných technologiích webových služeb.

3.3.3 XML Schema

Účel zavedení technologie XML Schema se váže k samotné podstatě XML. Tento jazyk umožňuje definovat svůj vlastní typ dokumentu včetně hierarchie zanoření uzlů, významu a hodnot elementů i atributů. Tato volnost přináší mnoho výhod, musí však existovat prostředek, který dokáže definovat omezení a proti těmto omezením jednotlivé dokumenty kontrolovat. Zmíněným prostředkem je právě XML Schema.

XML Schema je popis XML dokumentu, vyjádřený seznamem omezení aplikovaných na strukturu i její obsah. Narozdíl od DTD tak umožňuje kontrolovat i hodnoty dat jednotlivých prvků. Proces ověřování dokumentu se nazývá validace a dokument splňující všechna omezení je poté označen za validní. Možností XML Schema je ve webových službách využíváno hlavně k definici a kontrole obsahu SOAP zpráv.

3.3.4 Využití jazyka XML ve webových službách

Jazyk XML tvoří základy, na kterých lze stavět webové služby. XML poskytuje formát pro popis, ukládání i přenos dat. Neméně významným přínosem XML, využitým v rámci webových služeb, je definice nových jazyků. XML se stává základem, který je využit v mnoha oblastech.

První z nich je samotný popis služeb, prováděný WSDL dokumenty. Jazyk WSDL užívá jako jádro XML. Definuje vlastní sadu elementů určených pro konstrukci popisu rozhraní služby. Dalším příkladem je formát zasílaných zpráv mezi službami. Zde přichází na řadu protokol SOAP postavený právě na jazyce XML. Vyjímkou nejsou ani registry služeb UDDI nebo rozšíření jádra webových služeb - součásti rodiny standardů WS-*.

3.4 Popis webových služeb

3.4.1 Základní informace

Jazyk WSDL[1] (Web Services Description Language), jak už jeho název dostatečně napovídá, slouží k popisu webových služeb. Popis je určený pro případné spotřebitele služeb. Udává jim ty nejzákladnější informace zahrnující:

- metody, nabízené poskytovatelem
- protokoly pro navázání komunikace
- umístění poskytovatele

Hlavní část popisu webové služby by se dala do jisté míry srovnat s rozhraním používaným v Javě. Tak jako Java v rozhraní používá hlavičky metod, WSDL definuje tzv. koncové body. U každé metody definuje vše potřebné pro její správné zavolání. To znamená, že mimo názvu metody samotné také počet, jména a datové typy vstupních parametrů. Narozdíl od javovského rozhraní však povoluje definovat jeden či více výstupních parametrů spolu se jmény a datovými typy. Umístění poskytovatele je uváděno jako adresa zadávaná ve formátu URL. Pro navazování spojení se v popisu nachází typ komunikačního protokolu.

[1] WSDL ver. 1.1 (květen 2009), dostupný na URL http://www.w3.org/TR/wsdl

3.4.2 Struktura WSDL dokumentu

Služba, reprezentovaná množinou koncových bodů, je popsána v rámci XML struktury WSDL dokumentu. Pro ilustraci je v příloze B uveden vzorový WSDL dokument (přejato z [7]). [6] rozděluje popis na dvě na sebe navazující části:

- *Abstraktní popis* - Tato část dokumentu se zaměřuje na popis služby na určité úrovni abstrakce. To umožňuje definovat parametry služby, aniž by byla definice vázána na konkrétní implementaci. Popis je tak nezávislý na použitém operačním systému nebo způsobu komunikace. Do abstraktní části patří elementy:
  - `types` - tento element slouží k definici nových datových typů, které jsou určeny k definici parametrů metod nebo obecně k definici prvků dokumentu přenášeného v těle zprávy. Zde k vytváření nových datových typů slouží XML Schema. Element types není povinný, zapisuje se jen v případě existence nových datových typů.
  - `message` - obsah message záleží na preferovaném stylu komunikace. V případě stylu *RPC* obsahuje definici vstupních a výstupních parametrů volaných metod. U stylu *Document* nese informaci o zasílaném obsahu zprávy. Součástí elementu message bývá také definice chybových zpráv, které se zasílají v případě výskytu chyby v průběhu zpracování SOAP zprávy.
  - `portType` - tento prvek má k definici javovského rozhraní nejblíže. Definuje seznam metod služby, každou uzavírá do elementu `operation`. Výše připravené tagy message vkládá pro určení vstupních a výstupních parametrů metod.
- *Konkrétní popis* - Tento popis svazuje abstraktní část s konkrétní implementací závislou na aplikačním prostředí spolu s komunikačním protokolem pro přenos dat. Struktura obsahuje základní dva elementy:
  - `binding` - binding svazuje abstraktní definici rozhraní s konkrétním protokolem. Tím může být SOAP, HTTP nebo například MIME. Uvádí se také další informace vyžadované daným protokolem.
  - `service` - obsahuje seznam prvků `port` a každý z ních spojuje element `binding` s adresou, na které je služba dostupná.

Celá struktura zahrnuje komplexní popis webové služby. Existují nástroje, které dokáží celý popis vygenerovat na základě naimplementované služby. [5] ji však nedoporučuje z důvodu přímé svázanosti implementace s rozhraním a možností vzniku problémů při následné změně nástroje pro převod nebo implementace samotné.

3.5 Komunikace mezi webovými službami

3.5.1 Úvod

Protokol SOAP (Simple Object Access Protocol) je jednou ze základních specifikací. Zdroj [1] dokonce soudí, že je tou nejzásadnější ze všech, protože umožňuje aplikacím to nejpodstatnější a tím je výměna zpráv po síti. Protokol je založen, jako téměř vše ostatní, na jazyce XML, nezávisí tedy na použité platformě. XML obsah je na obou stranách komunikace zpracován SOAP procesorem, kdy je převáděn na nativní formát dat zpracovatelný danou implementací služby.

SOAP zprávy jsou samy přenášeny jako obsah zpráv jiných síťových protokolů. Nejčastějším způsobem, jak přenášet SOAP zprávu, je použití protokolu HTTP (Hypertext Transfer Protocol). Přenos pak funguje podobným způsobem, jak zasílání HTML dokumentů v případě Webu. Mimo přenosu přes HTTP lze dosáhnout stejného cíle použitím SMTP (Simple Message Transfer Protocol) originálně využívaného k emailové korespondenci nebo například protokolu pro přenos souborů FTP (File Transfer Protocol). Hlavní výhodou transferu přes HTTP je jeho rozšířená podpora díky Webu a fakt, že přenos na portu 80 v případě nechráněného HTTP a portu 443 u šifrovaného HTTPS není implicitně blokován firewally. Není tak bráněno masivnímu rozšíření webových služeb.

Komunikace HTTP protokolem probíhá zasíláním SOAP zprávy uvnitř HTTP požadavku a následným přijetím odpovědi od HTTP serveru. Podmínkou správné funkcionality je, že prvek zpracovávající HTTP zprávu musí být schopen ověřit správnost vnitřního obsahu SOAP zprávy podle uvedené specifikace.

3.5.2 Módy zasílaných zpráv

Mimo případu chybové zprávy nedefinuje SOAP, co má být obsahem těla zprávy (element `body`). Pokud je obsah správně formovaný podle syntaktických pravidel, může být tvořen čímkoliv včetně prázdného dokumentu. Jediným upřesněním zůstává mód zpráv. Ten kombinuje typ zprávy (*RPC, Document*) a kódování (*Encoded, Literal*).

RPC (Remote Procedure Call) typ říká, že zpráva obsahuje volání metod včetně zadaných parametrů. Naopak *Document* označuje obsah těla zprávy za XML strukturu. Styl kódování *Encoded* říká, že je zpráva kódovaná SOAP kódováním. U typu *Literal* se jedná o klasické XML validovatelné za pomocí XML Schema.

3.5.3 Struktura zprávy

Struktura SOAP zprávy je narozdíl od popisu služby WSDL dokumentu jednoduchá. Pro představu je zobrazena v dodatku C vzorová SOAP zpráva (přejata z [7]). Kořenový element `Envelope` slouží k zaobalení celé zprávy. V rámci jeho atributů se definují jmenné prostory použité uvnitř zprávy. Jinak zprávu rozdělujeme na dvě hlavní části:

- Hlavička zprávy (tag `Header`) - Hlavička slouží k metainformacím pro zpracovávání zprávy. Při cestě od odesilatele k příjemci prochází zpráva přes prostředníky (zvané také uzly), které ji čtou. Mohou modifikovat hlavičku, nikoliv však tělo zprávy. Mohou zde být uvedeny tzv. role, které musí daný uzel splňovat, aby mohl zprávu zpracovat. Existuje i možnost podmínit plnění role a při nesplnění reagovat chybovou zprávou směrem zpět k odesílateli namísto přeposlání zprávy dále. Hlavičky SOAP zpráv jsou také místem, kam se uvádějí informace pro zpracování rozšířenými webovými službami. Jedná se o prvky rodiny WS-* a označujeme je také pojmem „kvality služby“ (z angl. QoS - Qualities of Service). Detaily o rozšíření webových služeb jsou zmíněny v jedné z následujících sekcí. Hlavička není povinná, může být úplně vynechána.

- Tělo zprávy (tag `Body`) - Tělo SOAP zprávy obsahuje zasílaná data. Jimi je celý XML dokument nebo struktura pro volání metod služby v případě módu RPC. Obsahem těla zprávy může být také chybová zpráva. Ta je vložena při nenadálé chybové události a slouží k identifikaci příčin selhání.

3.6 Registry webových služeb

S postupným přechodem podniků na systémy postavené na architektuře orientované na služby narůstá také množství veřejně dostupných služeb. Aby se dalo využít jedné z výhod architektury - znovupoužití již naimplementované služby a zabránit tak opakovanému sestrojování stejné funkcionality, musí být vývojáři informováni o existenci dané služby. To může být problém, protože každý podnik uveřejňuje své služby na jiné adrese a neexistuje tak nikdo, kdo by měl přehled o tom, co je a co není naimplementováno. Řešení této situace nabízí UDDI (Universal Description Discovery Integration).

UDDI vzniklo na základě iniciativy firem IBM a Microsoft. Později se dostalo pod správu organizace OASIS, která UDDI přijala za svůj standard. UDDI zastává funkcí centrálního registru služeb. Slouží tedy jako místo, kde se poskytovatelé uveřejňují své služby a kam se obrací spotřebitelé, kteří hledají služby specifických funkcí. UDDI může zastávat jak funkci veřejného seznamu, tak i vnitropodnikové databáze služeb. Popisy služeb jsou uveřejněny v standardním formátu WSDL souboru. UDDI implementuje rozhraní pro SOAP zprávy a tudíž funguje sám o sobě jako webová služba. Zájemci kontaktují registr s žádostí ve formě SOAP zprávy. Výsledek je vrácen stejnou formou.

Registr poskytuje informaci o tom, co každá služba nabízí a kde je umístěna. Další informace se týkají organizace nabízející službu. Uvnitř registru je vše členěno do rozličných kategorií, podle kterých mohou zájemci vyhledávat. Mezi nimi jsou použité technologie, průmyslový obor nebo kategorie podniku a jiné.

Jak uvádí [5], původní idea registru však nebyla kompletně naplněna. Díky absenci kontroly jednotlivců, kteří služby přidávají, a dat, jenž zapisují, se registr postupně zaplnil nesprávnými a zmatenými údaji. Vyhledávání odpovídajících výsledků se tak stalo obtížným a neefektivním. Hlavní zastánci technologie UDDI firmy Microsoft, IBM a SAP ohlásily vypnutí svých veřejných registrů na počátku roku 2006.

3.7 Rozšíření webových služeb

3.7.1 Úvod

Jádro Webových služeb stojí na standardech jazyků XML a WSDL spolu s komunikačním protokolem SOAP. Zajišťuje základní popis služeb a definici způsobu vzájemné komunikace. V mnohých případech plnohodnotně postačuje k realizaci projektu založeného na architektuře orientované na služby. Ovšem na místech, kde se pracuje s citlivými daty, kde je vyžadováno výrazné zabezpečení dat nebo kde by chyba v systému mohla způsobit velké škody je potřeba dalších specifikací. Tyto prvky máme zvykem označovat jako *kvality služby* (z angl. QoS - Qualities of Service), protože se často starají o hůře implementovatelné oblasti prostředí SOA a přibližují tím webové služby blíže celkové myšlence architektury orientované na služby. Při definici jádra služeb se počítalo s potřebou jeho pozdějšího rozšiřování, byly tak připraveny specifické body pro jejich připojení. Jedním z nich je například hlavička SOAP zpráv.

Při vývoji nových specifikací se výrobci zaměřují ve většině případů na oblasti zlepšení bezpečnosti, zajištění transakčního zpracování operací nebo také spolehlivého doručování zasílaných zpráv. Pro zvýraznění náležitosti k webovým službám jsou specifikace označovány předponou WS a spadají tak do rodiny WS-*.

3.7.2 Správa Metadat

Správa metadat je podstatnou součástí webových služeb. Metadata slouží k definici pravidel určujících správný postup při konstrukci zprávy. Dodržování tohoto postupu umožní bezproblémové zpracování zprávy příjemcem. V případě jednoduchých služeb potřebuje spotřebitel získat informace o struktuře zprávy, adresu poskytovatele a vzor výměny zpráv. U služeb používajících rozšíření je navíc nutné získat metadata o politikách těchto specifikací.

[2] zmiňuje následující specifikace metadat:

- *XML Schema* - definice struktur a datových typů zprávy
- *WSDL* - spojování zpráv, způsobů jejich výměny
- *WS-Addressing* - adresace koncových bodů služeb a jejich vlastností
- *WS-Policy* - definice požadavků rozšířených webových služeb spojená s WSDL popisem
- *WS-MetadataExchange* - získávání metadat služby spolu s definicí WSDL a WS-Policy

Adresace koncových bodů umožňuje směrovat zprávy sítí. Definuje informace o tom, odkud je zpráva zaslána, kam směřuje a na který koncový bod má být zaslána odpověď. Poskytuje možnost zpracování hlaviček zprávy uzly po cestě mezi zasílatelem a finálním příjemcem. V případě výskytu chyby označuje koncový bod, kam se má zpráva vrátit. WS-Addressing je vyžadováno mnoha jinými specifikacemi.

WS-Policy doplňuje základní popis služby WSDL dokumentem o informace týkající se dalších požadavků poskytované služby. Zatímco ve WSDL dokumentu jsou definovány podmínky pro základní strukturu zprávy, WS-Policy specifikuje jaká rozšíření služba používá a jaké úpravy hlavičky jsou proto zapotřebí.

WS-MetadataExchange slouží jako prostředek pro přímé získávání dat. Poskytuje nástroje pro získání WSDL i s definicí politik od poskytovatele služby. Nahrazuje tak funkci veřejných UDDI registrů, které samy neobsahují správu metadat schopnou nabízet informace o politikách rozšíření.

3.7.3 Bezpečnost

Zajištění bezpečnosti je prioritou v mnoha oblastech informačních technologií. U webových služeb je potřeba připravit se na možné hrozby týkající se obecně distribuovaných systémů. Základní mechanismy pro zajištění bezpečnosti fungují na více úrovních a týkají se hlavně šifrování, autentifikace a autorizace.

První mechanizmus je aplikován na úrovni transportní vrstvy a váže se na použití HTTP protokolu. Jedná se o použití šifrovaného a ověřeného spojení za pomocí SSL. Komunikace ve formě HTTPS přináší výhody ve formě vysoké rychlosti zpracování a široké podpoře získané dlouhodobým používáním. Nevýhodou je pouze spojení z jednoho do druhého bodu, kdy neexistuje podpora zpracování uzly po cestě.

Druhou cestou je zabezpečení na úrovni SOAP zpráv samotných. Je třeba si uvědomit, že technologie SOAP je založena na XML a lze tedy využít bezpečnostní prvky určené právě pro XML. Ty mohou chránit jak data zprávy tak i její metadata:

- *XML Encryption* - Navržen pro zajištění důvěrnosti použitím mnoha podporovaných šifrovacích algoritmů. Umožňuje šifrovat kompletní XML dokument nebo pouze jeho část. Znesnadňuje tak případnému útočníkovi získání obsahu zprávy.

- *XML Signature* - Definován pro ustanovení integrity přenášených zpráv použitím algoritmů pro šifrování a podepisování. Ubezpečuje tak příjemce, že zpráva nebyla cestou pozměněna a že byla doručena pouze jednou.

V oblasti zabezpečení SOAP zpráv byla vydána specifikace *WS-Security*, která definuje pravidla pro používání šifrovacích a podepisovacích algoritmů. Hlavičky WS-Security umožňují autentizaci pomocí Kerberos nebo X.509 tokenů nebo zapojení XML Signature a XML Encryption pro další posílení bezpečnosti.

WS-Security muže být doplněna dalšími specifikacemi a tím i zvýšena její účinnost. Ty se zabývají zveřejňováním celkových požadavků vyžadovaných službou (WS-SecurityPolicy), zajištěním důvěryhodného sezení bez nutnosti opakovaně se autentizovat (WS-Trust) nebo například ustanovením sezení přes několik bezpečnostních domén (WS-Federation).

3.7.4 Spolehlivé doručování zpráv

V tomto odvětví se vývojáři zaměřují na vytvoření standardů, které mají garantovat správnou výměnu zpráv. Vzhledem k použití relativně nespolehlivého protokolu HTTP, je jejich přídavná funkcionalita nezbytná. Jedná se hlavně o to, aby byly zprávy stoprocentně doručeny, byly seřazeny ve správném pořadí a nevyskytovaly se žádné duplikáty. Stejně jako v jiných odvětvích i zde se objevuje více specifikací od různých tvůrců.

- *WS-ReliableMessaging* původem od IBM a Microsoft

- *WS-Reliability* z dílny OASIS

Ač nejsou specifikace totožné, jejich shodným účelem je výše zmíněná funkcionalita. Obstarávají je za pomoci číslování zpráv, jejich seskupování a řazení do sekvencí. Definují funkcionalitu, která by jinak musela být aplikační logikou.

Zasílání zpráv se týkají i specifikace *WS-Notification*, která byla ohlášena jako sloučení dvou konkurenčních větví:

- *WS-Notification* prosazovaná IBM a HP

- *WS-Eventing* s podporou Microsoftu a Intelu

Zde se jedná o podporu odlišných vzorů pro výměnu zpráv (z angl. MEP - Message Exchange Patterns) protokolu SOAP. Funguje pak broadcastové rozesílání zpráv nebo upozorňování na události přihlášeným odběratelům.

3.7.5 Transakční zpracování

Transakce slouží ke slučování jednotlivých operací do skupin za účelem vytváření určitých funkčních jednotek. Ty jsou důležité pro vykonávání vzájemně souvisejících operací. Podstatné je, aby se provedla celá transakční jednotka - tzn. všechny její části nebo se nevykonala vůbec. V praxi to znamená, že pokud se nezdaří jedna z operací v transakci díky hardwarové či softwarové chybě, transakční koordinátor je schopen vrátit vše do původního stavu. Tato vlastnost hraje klíčovou roli v transakčních prostředích.

Podpora transakcí může být dostatečná na úrovni aplikační logiky, některé aplikace však mohou vyžadovat provádění více webových služeb v rámci jedné transakce. U webových služeb se podpora transakcí provádí formou rozšíření transakčních koordinátorů spolupracujících aplikačních prostředí, úpravou dvoufázového potvrzovacího protokolu a definicí nových transakčních protokolů určených speciálně pro služby a jejich vzájemná spojení.

Proces koordinace transakcí probíhá ve dvou fázích. V první z nich jsou zúčastněné webové služby spojeny koordinátorem s konkrétním protokolem a ve druhé koordinátor řídí použitý protokol k provedení transakce. Při chybě koordinátor řídí operace zpět k návratu do původního stavu.

V rámci webových služeb byly vytvořeny tyto dvě skupiny specifikací:

- *WS-Transactions family* od IBM, Microsoft, BEA
- *WS-Composite Application Framework (WS-CAF)* od OASIS

Obě skupiny specifikací se zaměřují na rozšířené koordinátory transakcí s připojitelnými transakčními protokoly. Obě také definují atomické transakce a transakce založené na kompenzaci.

3.7.6 Orchestrace služeb

Síla aplikací postavených na SOA se ukazuje při vzájemné kooperaci jednotlivých služeb. V jednoduchých případech mohou služby volat samy sebe, ale pro vytvoření komplexnějších vzorů spolupráce je nutno zapojit další prostředky. Jedním z nich je orchestrace služeb. S pomocí orchestračního mechanismu je možné kooperujícími službami modelovat složitější toky podnikových procesů, používat konvenční prostředky jakými jsou větvení toků, vyvolávání chybových vyjímek nebo paralelního souběhu služeb. Při orchestraci je vytvořen jeden centrální prvek, který řídí tok celé orchestrace a navenek funguje jako nově vytvořená služba. Zahrnuté webové služby nejsou nijak speciálně upravovány a samy neví, že jsou součástí celého procesu.

V oblasti orchestrace se tvůrci specifikací služeb shodli na jednotném standardu, jímž se stal *WS-BPEL (Web Services Business Process Execution Language)*. Předpokladem použití WS-BPEL je popis služeb definovaný s WSDL a existence definic politik rozšíření. Mimo schopnosti přijímat a odesílat zprávy mezi službami, obsahuje také prostředky pro ukládání mezivýsledků, větvení, opakované vykonávání ve smyčkách či paralelní běh více služeb.

Rozšíření WS-BPEL[2] se ve své podstatě liší od ostatních specifikací. Zatímco ostatní specifikace umožňují aplikovat prostředky běžně používané v jiných architekturách pomocí úprav hlaviček SOAP zpráv, WS-BPEL definuje úplně nový spustitelný jazyk.

3.8 Nedostatky webových služeb

3.8.1 Funkcionalita vs. jednoduchost

Snahou tvůrců technologie webových služeb jako jedné z implementací architektury orientované na služby je, jako u mnohých produktů, maximální rozšíření a získání co největší obecné podpory. K tomu, aby se tak stalo, byla zvolena cesta těžící z úspěchů webu.

[2]WS-BPEL ver 2.0 (květen 2009), dostupný na URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Základní příčinou, proč se web stal tak úspěšným, je jeho jednoduchost. Jeho jádro je tvořeno pouze dvěma standardy. Prvním z nich je dnes všudypřítomný značkovací jazyk HTML a druhým komunikační protokol HTTP. K uvedení do provozu webové technologie stačí jen existence serveru rozumějícího příchozím HTTP požadavkům a na straně klienta prohlížeč se schopností vykreslit HTML dokument. Není proto třeba zabývat se otázkami použitého operačního systému, výrobce webového serveru nebo typu prohlížeče na straně klienta.

Webové služby zkoušejí určitým způsobem kopírovat právě jednoduchost webu a přenést ji do oblasti distribuovaných IT systémů. Místo jazyka HTML využívají jeho následníka XML a v komunikačním protokolu se dokonce shodují. I když existuje možnost aplikace jiného transportního protokolu, ve skutečnosti se žádné alternativy HTTP nevyužívá.

Jako případ opačného přístupu k implementaci distribuovaného systému [2] uvádí technologii CORBA (Common Object Request Broker). Tvůrci CORBA prosadili myšlenku standardizace jádra včetně záležitostí týkajících se bezpečnosti, transakcí, ap. s definovaným jednotným rozhraním. Díky zajištění těchto oblastí se jim podařilo markantně se přiblížit myšlence celé SOA. Konečný neúspěch CORBA byl však důsledkem absence standardního komunikačního protokolu a celkové složitosti systému.

S cílem zachovat jednoduchost webových služeb definují základní standardy XML, HTTP, SOAP a WSDL pouze omezenou funkcionalitu. Ke kompletní implementaci architektury orientované na služby však nepostačují. K tomu, aby se tak stalo, je potřeba rozšířit jádro o dodatečné specifikace. Rozšiřováním standardů webových služeb se ale vytrácí původní jednoduchost a vzrůstá tím potřeba zapojení čím dál komplexnějších struktur a mechanismů.

3.8.2 Standardizace

Standardizací lze nazvat postup, kdy je nějaký proces nebo technologie včetně specifikace veškerých detailů uznána za platný standard. Obecně uznávaným se stává až v případě, kdy je sám původce standardu respektován v dané oblasti nebo je považován za autoritu zabývající se vydáváním standardů.

Na uznání standardů jádra webových služeb se dohodli všichni významní tvůrci, a nebyl proto problém s výraznějším rozšířením. Obtíže se objevují při definici nových specifikací a snaze o jejich uznání. Celý proces začíná většinou ustavením malého týmu v rámci společnosti, který vyvine specifickou technologii, zveřejní ji a nechá uznat jednou z organizací za standard.

Na poli webových služeb funguje mnoho standardizačních organizací. [2] zmiňuje těchto šest:

- *World Wide Web Consortium* (W3C - http://www.w3.org/)
- *Web Services Interoperability* (WS-I - http://www.ws-i.org/)
- *Organization for the Advancement of Structured Information Standards* (OASIS - http://www.oasis-open.org/home/index.php)
- *Internet Engineering Task Force* (IETF - http://www.ietf.org/)
- *Java Community Process* (JCP - http://jcp.org/en/home/index)
- *Object Management Group* (OMG - http://www.omg.org/)

Jednotlivé organizace jsou odpovědné za vydání několika standardů. Každý z nich je uznán pouze částí celého spektra producentů. Mnohé z konkurenčních standardů se snaží dosáhnout stejného cíle ale odlišnou cestou. Neexistuje centrální organizace, která by řídila vývoj nových specifikací a vydávala všemi uznávané standardy. Ve výsledku tak vzniká velké množství navzájem se překrývajících specifikací, které mají pouze menšinovou podporu. Vývoj nových technologií je tím rapidně zpomalen a snižuje se tak celkový potenciál webových služeb jako implementace SOA (zdroj [5]).

3.8.3 Spolupráce rozšiřujících specifikací

Rozšiřujících specifikací se týká i další nevyřešená vlastnost webových služeb. Jak již bylo výše zmíněno, základní koncept webových služeb je postaven na expanzi funkcionality jejich jádra přídavnými rozšířeními. Úpravy se většinou soustředí na zásahy do SOAP zprávy, konkrétně do její hlavičky. Aplikační logika zpracovávající zprávu musí být také přizpůsobena. Při vzniku protokolu SOAP se s těmito modifikacemi počítalo a nevznikají tak žádné potíže.

Problém může nastat, pokud se začnou kombinovat různá rozšíření. Nikde neexistuje politika, která by jasně definovala pořadí definice těchto prvků. Při příjetí zprávy tak není jasné, v jaké posloupnosti se mají záznamy SOAP hlavičky interpretovat. Vznikají tak další nejasnosti, které mohou výrobce IT systémů odradit od používání technologie webových služeb.

Kapitola 4

Jazyk Java a webové služby

4.1 Úvod

Jednou z hlavních předností webových služeb jako implementace SOA je uváděna jejich nezávislost na použité platformě i hardware. Nezávislost je zde však zaručena pouze u komunikační infrastruktury. To znamená, že standardy jazyků WSDL, XML a komunikačního protokolu SOAP na XML založeného zaručují uniformní prostředí pro výměnu zpráv mezi službami. Pro jednotlivé webové služby to neplatí, protože jejich implementace je vždy pevně svázána s konkrétní platformou.

Jednou z takových platforem je Java Enterprise Edition (Java EE). V rámci Java EE jsou připravena dvě hlavní aplikační rozhraní - Java API for XML Web Services (JAX-WS) a Java API for XML Binding (JAXB). Nahrazují dříve používaný kit Java Web Services Development Pack (JWSDP). Popis původního i nového přístupu následuje.

4.2 Implementace webových služeb

4.2.1 Java Web Services Development Pack

JWSDP je sadou platformy Java EE pro vývoj webových služeb. Obsahuje seznam aplikačních rozhraní, kde každé z nich se soustředí na zprostředkování nástrojů pro podporu jiné technologie. Jejich distribuce v rámci balíku JWSDP poskytuje komplexní řešení pro vytváření aplikací využívajících webových služeb. Mezi základní API náleží:

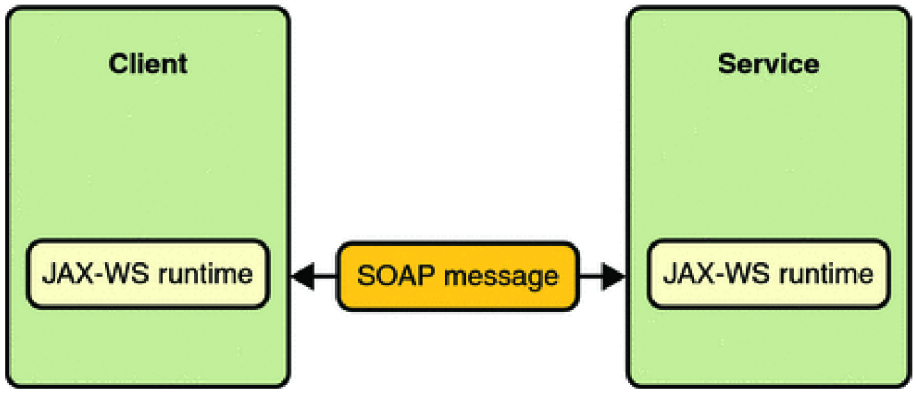
- *Java API for XML Processing (JAXP)* - aplikační rozhraní nabízející prostředky pro parsování XML dokumentů a jejich validaci prostřednictvím technologií SAX a DOM. Navíc podporuje transformaci dokumentů XSLT.
- *Java API for XML-based RPC (JAX-RPC)* - API pro volání vzdálených procedur. Podporuje RPC formát SOAP zpráv a popis jazykem WSDL.
- *Java API for XML Registries (JAXR)* - API pro přístup k registrům obsahujícím metadata služeb. Mezi podporovanými registry je i UDDI.
- *SAAJ (SOAP with Attachments API for Java)* - Aplikační rozhraní pro práci se SOAP zprávami včetně jejich příloh.
- *Java API for XML Messaging (JAXM)* - Rozhraní pro operace s asynchronními zprávami.

V roce 2006 skončila podpora balíku JWSDP ze strany výrobce, firmy Sun. Posledním vydáním je JWSDP 2.0. Nyní je většina technologii zahrnuta do nově vzniklého projektu Glassfish a pro tvorbu webových služeb byl zvolen odlišný, pro vývojáře jednodušší, styl.

4.2.2 Java API for XML Web Services

JAX-WS definuje nový způsob vytváření webových služeb a jejich klientů na platformě Java EE. Narozdíl od předchozího přístupu, kdy bylo odpovědností programátora zajistit základní funkcionalitu pomocí vhodně zvolené aplikace zmíněných rozhraní, zde je většina těchto operací schována použitím JAX-WS. Vše probíhá na vyšší úrovni abstrakce, jak ilustruje obrázek 4.1.

Zpětná kompatibilita JAX-WS je zaručena použitím služeb API z původního JWSDP. V rámci standardů webových služeb API podporuje jak dokumentový, tak RPC styl SOAP zpráv, jejich přenos v protokolem HTTP i popis služeb jazykem WSDL.



Obrázek 4.1: Abstrakce komunikace mezi klientem a službou (přejato z [16])

Služba

Webová služba je zapsána ve formě třídy. Narozdíl od normální definice jsou navíc přidány doplňující anotace. S pomocí anotací je později vygenerován ze zdrojového kódu třídy (tedy služby) odpovídající WSDL popis.

Klient

Komunikace klienta s webovou službou probíhá pomocí lokální proxy. Vytvoří ji programátor opět za pomoci anotací. Rozhraní proxy přesně kopíruje rozhraní třídy definující kontaktovanou službu a volání metod nad proxy tedy odpovídá volání metod služby. Veškerou logiku nutnou k realizaci komunikace obstarává JAX-WS API.

4.2.3 Java API for XML Binding

JAXB je aplikační rozhraní, které doplňuje JAX-WS API o technologie konverze dat. Definuje rychlý a jednoduchý způsob, jak navazovat XML Schemata na odpovídající reprezentace jazyka Java. A naopak je schopné z Java tříd generovat XML Schemata. Dokáže oběma směry převádět mezi sestavou XML dokumentů a odpovídající stromovou strukturou

v Javě. Od verze JAXB 2.0 se dá provádět i dodatečná validace XML struktur zapojením JAXP.

4.3 Java a SOA Governance

V rámci platformy Java Enterprise Edition jsou vyvíjeny nástroje pro práci s webovými službami. Výše zmíněná rozhraní poskytují obsáhlé a propracované prostředky k implementaci webových služeb a jejich integraci do rozsáhlejších aplikací. Podpora nástrojů pro prosazování SOA Governance není zdaleka tak rozsáhlá. SOA Governance se dá prohlásit za jednu z nejméně propracovaných oblastí celé architektury orientované na služby.

Jak již bylo diskutováno výše, pojem SOA Governance zahrnuje celou škálu domén působnosti náležících do architektury orientované na služby. Všechny mají jedno společné a tím je kontrola a řízení služeb po celou dobu jejich života. Podle fáze aplikování by se daly rozdělit na část formující pravidla v době návrhu služeb a na ostatní definice týkající se doby provozu.

Právě do druhé skupiny, můžeme zařadit GlassFish server. GlassFish je open-source Java EE 5 aplikační server vyvíjený v rámci projektu GlassFish[1]. Tento server je šířen samostatně nebo v rámci platformy Netbeans a poskytuje ideální nástroj pro provozování webových služeb implementovaných nad Java EE. Z pohledu SOA Governance je zajímavý svým nástrojem pro monitorování služeb běžících na serveru.

GlassFish umožňuje podle [12] kontrolovat webové služby pomocí sady nástrojů sestávajících se příkazové řádky (tzv. CLI - command-line interface), administrační konzole (s vlastním GUI) a programovatelných aplikačního rozhraní Application Server Management Extensions (AMX). Tomuto rozhraní se věnuje kapitola 7. Pro demonstraci funkce jsou uvedeny možnosti administrační konzole. Nabídka je rozstrukturována do čtyř záložek:

- *Obecné* - umožňuje výpis veškerých služeb umístěných na serveru. Pro každou z nich pak zobrazuje základní informace o názvu, URI koncového bodu, typu implementace a jiné. Připojuje také dokument WSDL. Za účelem otestování správné funkce generuje jednoduchou testovací proceduru.

- *Monitorování* - dovoluje nastavit úroveň monitorování každé služby. Volnější nastavení zaručuje pouze sbírání statistik, plné monitorování přidává možnosti stopování zpráv služby. Statistiky zahrnují časové údaje o zpracování požadavků a indikaci úspěšných a chybných žádostí. U zpráv se uvádí adresa zdroje, jméno uživatele, datum a čas volání, typ transportního protokolu. Dostupné jsou i obsahy žádostí a odpovědí včetně jejich velikostí. Všechna data jsou filtrovatelná podle úspěšnosti odbavení. U časových statistik jsou k dispozici grafické interpretace.

- *Transformace* - zobrazuje dostupné transformace přes XSLT. Ty je možno aplikovat na různé koncové body a způsobit tím konverzi XML obsahu všech zpráv s konkrétním koncovým bodem spojených.

- *Publikování* - slouží k vytváření a k přístupu k registrům webových služeb. Jsou podporovány standardy ebXML a UDDI. Po ustavení registrů lze volně publikovat nové webové služby. Pro vyhledávání služeb v registrech je připraveno rozhraní.

[1] Více o projektu GlassFish na adrese https://glassfish.dev.java.net/

Kapitola 5

Technologie JMX - Management Java aplikací

5.1 Úvod

S příchodem jazyka Java se objevila, stejně jako v případě jiných jazyků, potřeba prostředků pro správu aplikací v něm napsaných. V počátcích užívání se Java soustředila do oblasti spíše jednodušších lokálně umístěných krátkoběhových aplikací. Náročnost správy odpovídala přímo úměrně jejich nevelké složitosti. Proto se k těmto účelům s úspěchem využívalo pouze logovacích souborů, chybových konzolí nebo nástrojů poskytovaných hostitelským operačním systémem. S přibývajícím časem a zvyšujícími se možnostmi jazyka narůstala i složitost produktů na něm založených. Java platforma se dostala do oblasti podnikových systémů. Spolu se zaváděním aplikací založených na architektuře orientované na služby a obecně příchodem distribuovaných systémů přestaly jednoduché prostředky pro management stačit a objevila se potřeba komplexnějšího řešení.

Tématem managementu aplikací Java platformy se začala zabývat skupina, která dohlíží na vývoj jazyka a připravuje podněty k ustanovení nových specifikací - Java Community Process (JCP). Na základě žádostí jednotlivých členů komunity v podobě dokumentů Java Specification Request (JSR) vznikají specifikace nových i opravy a doplnění stávajících produktů. Výsledkem snahy JCP vznikla specifikace Java Management Extensions (JMX). Její první verze byla vytvořena na základě požadavku JSR003. Kompletní seznam žádostí souvisejících s JMX je v tabulce 5.1. Později bylo JMX producenty Javy přijato za standard a od verze Java SE 5.0 je JMX plnohodnotnou součástí distribuce. Aplikační rozhraní JMX je umístěno v balíku `javax.management`.

Označení	Obsah specifikace	Popis specifikace
JSR003	JMX verze 1.0 - 1.2	základní specifikace JMX
JSR160	JMX Remote API	vzdálený přístup k managementu
JSR255*	JMX verze 2.0	dodatečná funkcionalita
JSR262*	JMX Remote WS API	přístup k managementu přes WS

Tabulka 5.1: Tabulka žádostí o specifikace Java Community Process pro JMX. Položky označené * jsou v době psaní tohoto textu (květen 2009) před schválením.

Technologie JMX představuje architekturu řídících[1] systémů. Definuje vzory pro jejich návrh, aplikační rozhraní i služby pro monitorování a management na platformě Java. JMX byla podle předchozích požadavků navržena tak, aby bylo možné integrovat nově vzniklé JMX systémy do těch stávajících. Představuje cestu, jak jednoduše doplnit kód existujících aplikací novými prvky a zpřístupnit tak aplikace řídícím systémům.

Způsob úpravy kódu podle Java Management Extensions není náročný a lze jej aplikovat v širokém spektru oblastí Javy. Z tohoto pohledu se stává technologie JMX zajímavou i pro konstrukci a úpravu webových služeb. Implementace webové služby je tvořena jednoduchými Java objekty - POJO (z angl. Plain Old Java Object) doplněnými o anotace. Není tedy obtížně ji kdykoliv zpřístupnit managementu přidáním jen několika řádků kódu.

5.2 Architektura

Hlavním znakem architektury řídících systémů založených na JMX je rozdělení do tří základních vrstev. Obrázek 5.1 ilustruje strukturu typického systému. První z vrstev, která je označována jako instrumentační (z angl. instrumentation layer), obsahuje části spravovaných aplikací. V této vrstvě se provádí doplnění kódu zdrojových aplikací, která je zpřístupňuje řídícím systémům. Druhou částí je vrstva agentní (z angl. agent layer). Cílem agentní vrstvy řízené spojení spravovaných modulů a řídícím aplikací. Poslední vrstva je označována jako distribuované služby. Slouží k realizaci spojení JMX systému s řídící aplikací nezávisle na jejím umístění a použitém komunikačním protokolu.

5.2.1 Instrumentační vrstva

Instrumentační vrstva je významnou součástí celého systému. Obsahuje veškeré zdroje dat, které jsou určeny k managementu a monitorování. Probíhá zde jejich dodatečná instrumentace tak, aby je bylo možné řídit a spravovat. Do oblasti spravovatelných prvků můžeme zařadit vše od jednoduchých webových služeb přes komplexnější moduly až po celé aplikace.

MBeany

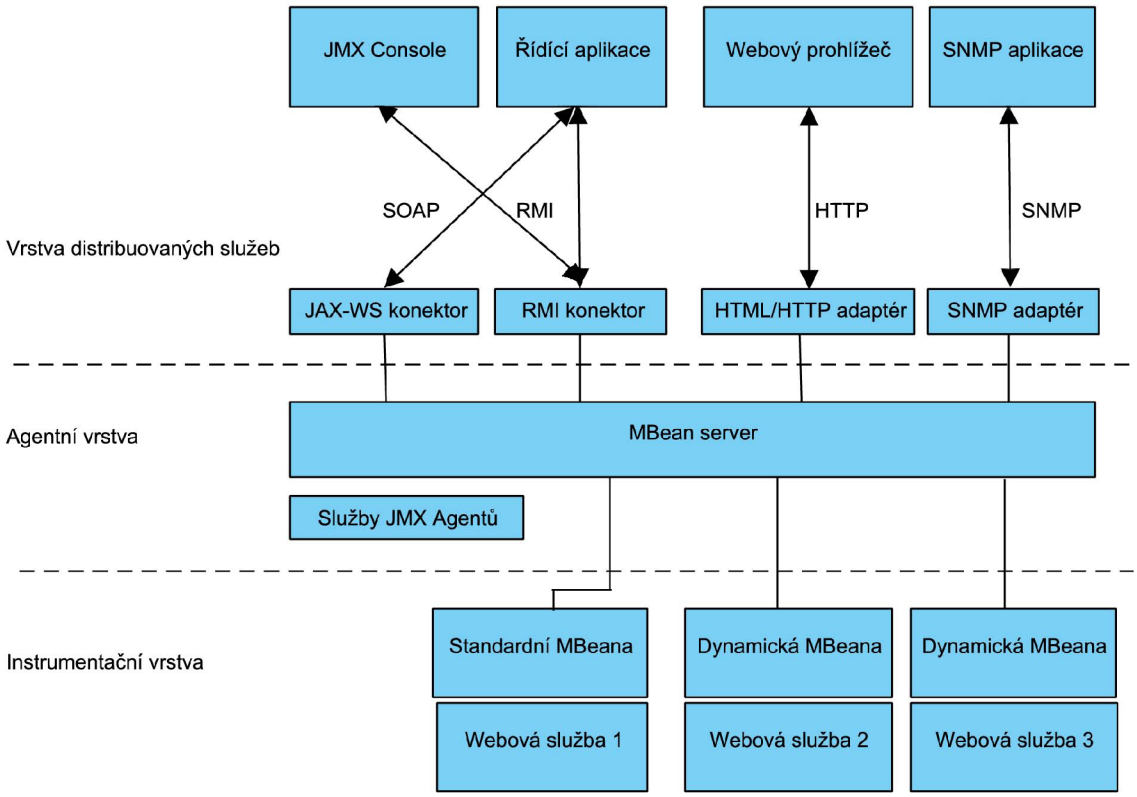
Systémy Java Management Extensions sjednocují přístup ke všem prvkům definováním společné jednotky managementu. V terminologii JMX je označována jako Managed Bean, zkráceně MBean[2]. Nezávisí tedy na druhu spravovaného prvku, protože na vše je v rámci systému pohlíženo jako na objekt typu MBean. Tento princip radikálně zjednodušuje správu všech prvků.

MBeanu lze definovat jako běžný Java objekt, jenž má několik omezení. Prvním z nich, které vychází z technologie JavaBean, je přístup k atributům objektu. Pro každý z atributů musí být specifikovány přístupové metody `set` a `get`[3] na základě jeho čitelnosti/zapisovatelnosti. Druhým požadavkem na objekt mbeany je použití řídícího rozhraní. Definicí tohoto rozhraní je jednoznačně určeno, co má být zpřístupněno ke správě a co má být naopak před ní skryto.

Obecně je rozhraní tvořeno těmito částmi:

- Atributy určující aktuální stav objektu

[1] Přívlastek *řídící* používáme pro objekty z oblasti managementu určené ke správě a řízení
[2] My budeme používat počeštěný termín *mbeana* a vyhneme se doslovným překladům
[3] V češtině se často pro zjednodušení označují jako *setter* a *getter*



Obrázek 5.1: Architektura JMX systémů

- Operace připravené pro volání řídící aplikací

Samotná identifikace vlastností mbeany je získávána z jejích metadat. Součástí metadat je kromě seznamu atributů a operací také definice konstruktorů a událostí, které je mbeana schopna zasílat. Podle způsobu a času konstrukce metadat rozdělujeme čtyři základní typy mbean:

- *Standardní mbeany* - vlastní definice rozhraní. Metadata jsou generována v době návrhu. Minimální objem napsaného kódu.
- *Dynamické mbeany* - dostupné rozhraní nutí programátora k implementaci všech metod pro generování metadat. Výhodou je flexibilita konstrukce metadat v době běhu programu.
- *Open mbeany* - dynamické mbeany s omezením použitých datových typů
- *Model mbeany* - dynamické mbeany implementovány připravenou třídou s možností úpravy. Snižují objem napsaného kódu.

Zpřístupňované prvky nejsou závislé na systému, jenž je využívá. Jejich instrumentace není složitá a nevyžaduje znalosti žádného konkrétního management systému. Ve většině případů se jedná pouze o doplnění několika řádků do původního zdrojového kódu. Po splnění výše zmíněných podmínek je prvek připraven ke správě.

5.2.2 Agentní vrstva

Agentní vrstva se nachází uprostřed mezi instrumentační vrstvou a vrstvou distribuovaných služeb. Skládá se z jednoho či více JMX agentů. Každý z nich obsahuje vždy dvě části - MBean server a služby JMX agenta. JMX agenti jsou ve většině případů lokalizováni na stejném virtuálním stroji jako mbeany, se kterými pracují. Těží tedy z výhody přímého přístupu.

MBean server

Hlavním úkolem každého MBean serveru je stále udržovat přehled o použitých prvcích. Server slouží jako registr mbean, které jsou zpřístupňovány řídícím aplikacím. Vše funguje na jednoduchém principu registrace. K tomu, aby mohla být mbeana použita, musí být nejprve vytvořena její instance a následně zaregistrována právě u MBean serveru. Podmínkou úspěšného provedení registrace je přiřazení jedinečného identifikátoru v rámci celého serveru. Dokud není mbeana registrována, nelze ji využívat.

Druhou a neméně podstatnou funkcí MBean serveru je zprostředkování přístupu management aplikacím k vybraným mbeanám. Server umožňuje aplikacím vyhledávat jednotlivé mbeany podle parametrů, pracovat s jejich atributy, vyvolávat operace a využívat systému zasílaných notifikací. Plní roli jistého delegáta a dovoluje tak aplikacím využít pouze prvků, které jsou k tomu určeny.

Služby JMX agentů

Spolu s MBean serverem tvoří součást JMX agenta také jeho služby. Každá z nich je realizována jako samostatná mbeana a její přítomnost je v agentovi povinná. Vztahují se na ni všechny výhody i nevýhody standardních mbean. Jde o následující čtyři typy služeb:

- *Monitorovací služby* - sledování hodnot atributů mbean, zasílání událostí založených na předem stanovených podmínkách
- *Služba časovače* - prostředek pro zasílání událostí na časové bázi
- *Služba class loaderu* - možnost nahrávat mbeany ze vzdálených umístění
- *Služba vztahů* - realizace vzájemných vztahů dvou a více mbean

5.2.3 Vrstva distribuovaných služeb

Součástí vrstvy distribuovaných služeb jsou mechanismy, které zajišťují propojení mezi vzdálenými řídícími aplikacemi a agentní vrstvou, konkrétně MBean serverem. Podle principu funkce se dají rozdělit do dvou skupin - na konektory a adaptéry protokolů.

Konektor

Hlavním účelem konektorů v JMX je zajištění transparentního spojení mezi řídící aplikací a JMX agentem. Konektor poskytuje aplikacím rozhraní, které zpřístupňuje MBean server nezávisle na jeho umístění. Při použití konektoru se neliší přístup k lokálnímu a vzdálenému MBean serveru.

Konektory jsou rozděleny na dvě samostatné části. Klientská část definuje proxy objekt MBean serveru, přes který se provádí veškeré operace nad mbeanami. Iniciuje také spojení

podle potřeb řídící aplikace. Serverová část naslouchá klientovi a podle jeho žádostí ustavuje spojení. Konektor zajišťuje obousměrný provoz mezi klientem a serverem.

V době psaní tohoto textu (květen 2009) je oficiálně dostupný pouze konektor RMI (Remote Method Invocation). Tento konektor vznikl na základě žádosti o specifikaci JSR160 (viz. tabulka 5.1) a po schválení specifikace stal součástí J2SE 5.0. Implicitně je kombinován s transportním protokolem JRMP (Java Remote Method Protocol), ale lze jej provozovat i ve spolupráci s protokolem IIOP (Internet Inter-ORB Protocol). Druhý konektor, který by měl rozšířit konektivitu JMX systémů, je založen na webových službách. Tímto tématem se zabývá JSR255, jež je ale v současné době stále ve fázi návrhu. Tyto informace vycházejí z [10].

Adaptér protokolu

Způsob využití adaptéru se od konektoru podstatně liší. Narozdíl od konektoru je každý adaptér tvořen pouze jedinou částí umístěnou vždy na straně serveru. Jeho úkolem je transformovat aplikační rozhraní MBean serveru do podoby schopné zpracování vybraným protokolem. Jako příklad protokolu, který lze využít ke spolupráci s JMX systémem, lze zmínit SNMP (Simple Network Management Protocol). S použitím tohoto adaptéru lze jednoduše integrovat prostředky pro správu přes JMX do již existujících řídících systémů využívajících služeb SNMP protokolu. Druhým příkladem může být HTML adaptér nad protokolem HTTP, s jehož pomocí lze zobrazovat obsah spravovaných objektů ve formě HTML stránek. V porovnání s konektory však není ani jeden z adaptérů uznán za standard.

5.3 J2EE Management Model

Specifikace Java Management Extensions definuje obecný způsob, jak má vypadat struktura řídících systémů Java platformy. Určuje, jakým způsobem přistupovat k instrumentaci spravovaných prvků nebo jak řídit přístup aplikací k nim. V žádné ze svých částí se ale nezabývá konkrétní reprezentací spravovaných prvků. Proto vznikla ve skupině JCP další návrh specifikace řešící tuto tématiku.

Výsledkem JSR77 se stala specifikace J2EE Management, která představuje společný model pro objekty J2EE platformy - tzv. J2EE Management Model. Tento model založený na JMX stanovuje, jak mají být reprezentovány spravované prvky - virtuální stroje Javy, EJB moduly nebo například servlety.

Hlavním znakem J2EE Management Modelu je definice třídy `J2EEManagedObject` jako společného základu. Všechny spravované prvky vychází z této třídy a dále jej rozšiřují o své specifické vlastnosti. Obsah třídy J2EEManagedObject ukazuje tabulka 5.2.

Atribut	Datový typ	Význam
`objectName`	OBJECT_NAME (String)	jednoznačná identifikace
`stateManageable`	boolean	možnost správy stavu objektu
`statisticsProvider`	boolean	poskytování statistik o běhu
`eventProvider`	boolean	zasílání událostí

Tabulka 5.2: Struktura J2EEManagedObject objektu.

J2EE Management Model představuje určité společné minimum pro modelování řídí-

cích systémů Java platformy. Díky jednotnému základu se unifikuje vývoj systémů pro J2EE servery různých výrobců a zjednoduší se jejich vzájemná integrace. Model také poskytuje ideální bázi, ze které mohou vycházet producenti J2EE serverů a rozšiřovat ji o své nástavbové prvky.

Kapitola 6

Správa a řízení mbean

6.1 Instrumentace webové služby

6.1.1 Úvod

Instrumentace programového kódu není nic jiného než jeho doplnění o prostředky managementu. S jejich pomocí je možné zahrnout webovou službu mezi prvky spravované a ovládané řídící aplikací. Rozsah instrumentace vždy závisí na množství informací zpřístupňovaných řídícímu systému. Ovládání webové služby pak probíhá přes řídící rozhraní služby definované procesem instrumentace.

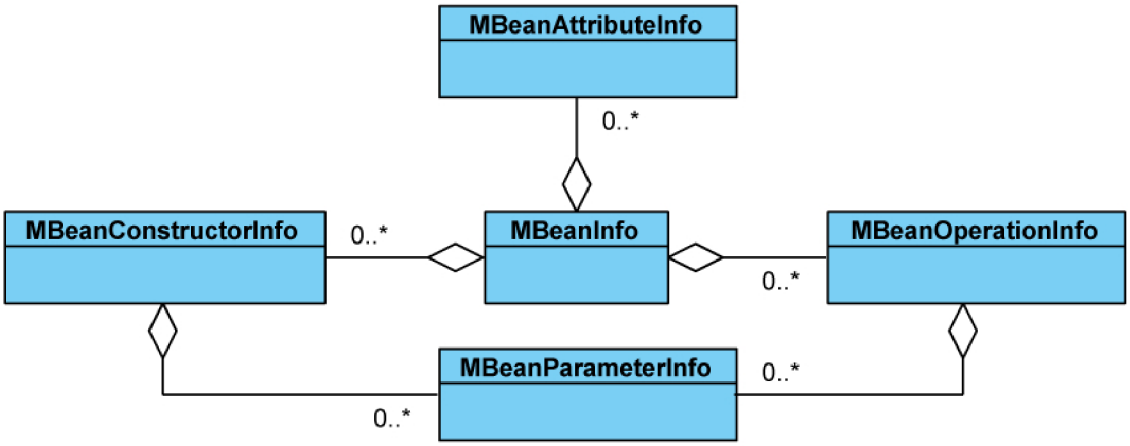
6.1.2 Metadata mbean

Ještě předtím, než můžeme popsat postup instrumentace webové služby, je potřeba představit, jakým způsobem je reprezentována struktura mbeany v celém řídícím systému. V případě konstrukce standardního typu mbean není programátor schopen tuto informaci ovlivnit, proto pro něj není moc podstatná. Ale v okamžiku, kdy se jedná o instrumentaci webových služeb dynamickou cestou, sestavuje metadata programátor sám a proto je pochopení struktury metainformací mbeany důležité.

Za účelem reprezentace řídícího rozhraní mbean existuje v GlassFish serveru speciální třída `MBeanInfo`. Tato třída sdružuje informace týkající se celého rozhraní. Její podtřídy reprezentují jednotlivé vlastnosti mbeany. Celková struktura metadat mbeany je zobrazena na obrázku 6.1.

Metadata mbean jsou tvořena následujícími třídami:

- `MBeanInfo` - Reprezentace kompletního řídícího rozhraní. Nejvyšší úroveň metadat.
- `MBeanAttributeInfo` - Definice atributů. Kromě názvu a datového typu jsou zde uloženy tři booleovské příznaky pro vyznačení možnosti čtení a zápisu do atributu.
- `MBeanOperationInfo` - Popis operací mbeany. Název a návratový typ jsou doplněny vyznačením dopadu volání operace. Tento prvek má informační charakter a určuje se na základě existence vstupních parametrů a vracené hodnoty. Posledním parametrem je tzv. signatura, která formou pole definuje vstupní parametry.
- `MBeanParameterInfo` - Třída pro reprezentaci parametrů operací a konstruktorů. Zde je spolu s názvem uložen i datový typ.



Obrázek 6.1: Schéma metadat mbean

- `MBeanConstructorInfo` - Veřejné konstruktory mbeany. Třída se strukturou shoduje s `MBeanOperationInfo`. Vyjímkou je absence návratového datového typu a vyznačení dopadu volání.
- `MBeanNotificationInfo` - Deskriptor notifikací. Určuje název a typ zasílané notifikace.

Společným znakem všech výše zmíněných tříd je existence vlastního textového popisu, jenž je určen k usnadnění pochopení prvku člověkem. Pro strojové zpracování nemá žádný význam.

6.1.3 Instrumentace webové služby standardní mbeanou

Použití standardního typu mbeany je nejjednodušší cestou jak zpřístupnit webovou službu managementu. Po tvůrci zdrojového kódu vyžaduje minimum úsilí. Jeho hlavním úkolem je definovat řídící rozhraní, u kterého platí stejná omezení jako při konstrukci normálního rozhraní jazyka Java. Je třeba ovšem splnit několik dodatečných podmínek.

Rozhraní je uvedeno jako veřejné a jeho název je odvozen od názvu třídy, která jej implementuje. Vzniká doplněním klíčového slova `MBean` za jméno třídy mbeany. Například:

```
public interface WebServiceMBean{...}
public class WebService implements WebServiceMBean{...}
```

Jak již bylo zmíněno v sekci 5.2.1, mbeany používají technologii JavaBean. To umožňuje kontrolovat jejich atributy přes řídící rozhraní definicí metod `set` a `get` s odpovídajícím vstupním parametrem nebo návratovým datovým typem. Podle dostupnosti těchto metod je určováno, zda můžeme atributy číst, zapisovat do nich nebo obojí. Chceme-li vystavit atribut `attribute` ke čtení i zápisu, musíme definovat metody `setAttribute` a `getAttribute`. V případě booleovského datového typu lze metodu `getAttribute` nahradit metodou `isAttribute`. Přístupové metody nelze přetěžovat.

Zde je třeba připomenout, že sestavování atributů je prováděno JMX agentem na základě setteru a getteru v řídícím rozhraní. Proto je nutná ostražitost při jejich psaní. Metoda s nesprávným parametrem, návratovým typem nebo odlišnou velikostí písmen je považována

za normální operaci. Agent kvůli chybějící přístupové metodě nemusí nakonec atribut vůbec zobrazit.

Pro definování řídících operací nejsou specifikovány žádné konkrétní podmínky. Vždy se uvádí pouze hlavička metody jako v případě standardních rozhraní Javy.

Při konstrukci rozhraní, stejně tak jako při jeho implementaci webovou službou lze využít výhody mechanismu dědičnosti. Pokud otcovská třída implementuje své řídící rozhraní a my ji zdědíme, rozhraní je automaticky aplikováno i na třídu synovskou. Jak je však uvedeno v [3], toto tvrzení neplatí, pokud synovská třída zároveň implementuje své vlastní řídící rozhraní. K tomu, abychom mohli využít kombinace rozhraní synovské i otcovské třídy, je nutné specifikovat dědění klíčovým slovem `extends` jak u synovské třídy, tak u jejího rozhraní. Příklad implementace vypadá následovně:

```
public interface ParentWSMBean{...}
public class ParentWS implements ParentWSMBean{...}

public interface ChildWSMBean extends ParentWSMBean{...}
public class ChildWS extends ParentWS implements ChildWSMBean{...}
```

Správná definice rozhraní standardní mbeany je velice důležitá, protože při registraci podle ní JMX agent konstruuje objekt metadat `MBeanInfo`. Proces procházení, kontroly a získávání informací se nazývá *introspekce* (z angl. introspection). Introspekcí řídícího rozhraní je agent schopen sestavit metadata o atributech i operacích. K vytvoření metadat konstruktorů je využita implementující třída. Podmínkou je existence nejméně jednoho veřejného konstruktoru. V případě, kdy třída zároveň sdružuje kód implementace webové služby i její mbeany, musí být uveden konstruktor bez parametrů.

Notifikační mechanismus je zajištěn implementací odlišného rozhraní. Této problematice je věnována samostatná sekce 6.1.5.

Třída webové služby musí být konkrétní, veřejná a je povinna implementovat všechny metody řídícího rozhraní. Odhalování nedostatků při definici rozhraní nebo jeho implementaci webovou službou je obtížné. I kód nedodržující zmíněná pravidla může být validní a přeložen bez problémů. Na chybu se pak přichází v okamžiku registrace mbeany při jejím prvním použitím. Introspekce JMX agentem neproběhne v pořádku a je vyvolána vyjímka.

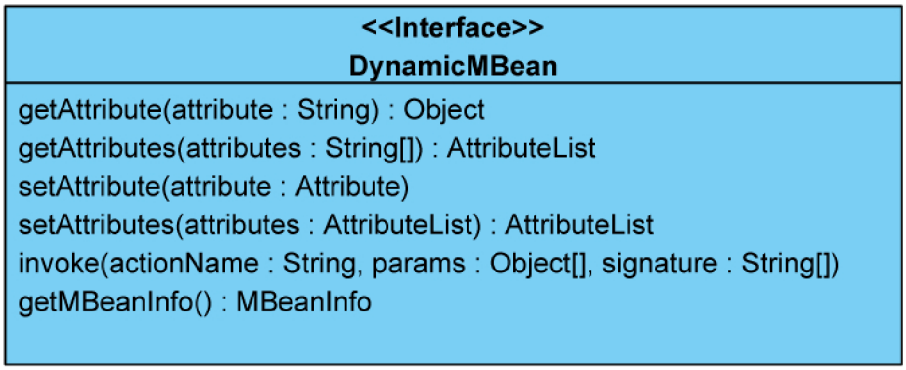
6.1.4 Instrumentace webové služby dynamickou mbeanou

Zatímco instrumentace webových služeb cestou standardní mbeany vyžaduje konstrukci vlastního řídícího rozhraní, pro dynamickou mbeanu tento krok není třeba. Namísto přípravy rozhraní specifického pro každou službu se využívá předem připraveného obecného rozhraní `DynamicMBean` (schéma na obrázku 6.2).

V případě dynamické mbeany se neprovádí proces introspekce pro získaní informací o zveřejněných parametrech. Vytvářením metadat dynamické mbeany je pověřen programátor právě implementací `DynamicMBean`. Konstrukce objektu `MBeanInfo` se provádí na žádost JMX agenta až v době běhu aplikace. Struktury přístupné managementu se mohou měnit v průběhu času. Proto při sekvenci žádostí o objekt `MBeanInfo` mohou být výsledkem odlišné hodnoty. Z tohoto důvodu je typ mbeany označován jako dynamický.

Implementací rozhraní `DynamicMBean` zajistí programátor kód těchto metod:

- `getAttribute` - jednoduché získání hodnoty atributu podle zadaného jména. Samozřejmostí je kontrola existence požadovaného atributu a jeho přístupnost ke čtení.
- `getAttributes` - shodné s předchozím. Pracuje se s více atributy najednou.



Obrázek 6.2: Definice rozhraní DynamicMBean

- `setAttribute` - nastavení hodnoty vybraného atributu. Součástí musí být kontrola existence atributu, jeho přístupnost k zápisu a ověření nastavované hodnoty z hlediska datového typu i povoleného rozsahu hodnot.
- `setAttributes` - shodné z předchozím. Nastavuje se více atributů najednou.
- `invoke` - metoda zprostředkovává volání operace řídícími aplikacemi. Podle jména volané operace se musí ověřit její dostupnost. Spolu se jménem operace je na vstupu předána signatura a pole objektů reprezentující hodnoty vstupních parametrů operace. Po ověření validity vstupních parametrů je nutné zajistit vyvolání metody a poté zpětné zaslání návratové hodnoty, pokud existuje.
- `getMBeanInfo` - metoda nahrazující proces introspekce. Správné sestavení metadat je odpovědností programátora. Struktura musí odpovídat schématu na obrázku 6.1. Při konstrukci metadat se nesmí opomenout definice konstruktorů třídy a identifikace případných notifikací.

Také v případě dynamické mbeany lze využít mechanismu dědičnosti. Pro získání metadat mbeany se používá nejbližší možná implementace metody `getMBeanInfo` v hierarchii dědičnosti.

Při detekování jakéhokoliv pochybení ze strany klienta mbeany je vhodné využít mechanismu vyjímek jazyka Java. Spolu se předdefinovanými třídami vyjímek můžeme aplikovat i své vlastní, specifické. Uvedením klíčového slova `throws` v hlavičce metody propagujeme vyvolanou vyjímku přes MBean server zpět k původci volání.

Velká volnost při definici dynamické mbeany má své klady i zápory. Je v rukou programátora, jak implementuje veškerou logiku i jak ošetří případné výskyty chyb. Kladem je také možnost nastavení čitelného a logického popisu všech prvků metadat.

Hlavním záporem při instrumentaci webové služby touto cestou je nezbytnost obstarání téměř veškeré funkčnosti mbeany. S tím souvisí velký objem napsaného kódu pro každou službu a následné riziko výskytu chyb.

6.1.5 Notifikační model JMX

Instrumentací webové služby formou implementace rozhraní získávají řídící aplikace komplexní prostředky pro kontrolu mbean. Samotné rozhraní však slouží pouze jako pasivní

nástroj pro přístup k informacím. K tomu, aby mohla řídící aplikace reagovat na splnění předem definované podmínky uvnitř zpřístupněného prvku nebo aktuální změnu jeho stavu, je zapotřebí doplnit mbeanu o schopnost aktivního chování.

Za tímto účelem byla specifikace JMX vybavena takzvaným notifikačním modelem založeným na obecném modelu událostí Javy. Stejně jako v případě obecného modelu i zde jde o zasílání událostí ve formě notifikací mezi jednotlivými moduly systému. Odlišujeme dvě základní role prvků - vysílač událostí a jejich příjemce (většinou označovaný jako posluchač). Princip zasílání událostí funguje navzájem mezi všemi mbeanami nezávisle na jejich umístění. Každá z mbean se může stát současně vysílačem událostí i jejich příjemcem. Řídící aplikace může hrát roli posluchače.

Způsob použití notifikačního modelu JMX u mbeany webové služby se nijak neliší od postupu u mbeany normální Java třídy. Pokud chceme využít služeb notifikačního modelu u mbeany webové služby, je třeba postupovat podle následujících kroků.

Příprava mbeany vysílající události

K tomu, aby se z webové služby stal zdroj událostí je nutné implementovat jedno z rozhraní `NotificationBroadcaster` a `NotificationEmitter`. Druhé v pořadí můžeme označit za modernější verzi prvního. Zařazením jednoho z rozhraní poskytujeme posluchačům způsob jak se registrovat a odregistrovat z příjmu událostí.

Při implementaci rozhraní můžeme postupovat čistě vlastní cestou. Rozumnějším řešením je však využití služeb jedné ze tříd `NotificationBroadcasterSupport` (součást J2SE 5.0) nebo `NotificationEmitterSupport` (záležitost pouze GlassFish serveru), jež rozhraní samy implementují. Pokud nelze aplikovat dědění, můžeme zapojit třídu jako členskou proměnnou mbeany a ve vlastních metodách na ni delegovat veškerá volání.

MBeany produkující události obsahují seznam zaregistrovaných posluchačů, který slouží jako adresář. Událost je odesílána postupně na všechny z nich.

Příprava posluchače

Proměna jakékoliv třídy v posluchače událostí je definována podobným způsobem. Rozhraní `NotificationListener` se svou jedinou metodou `HandleNotification` určuje prostor v posluchači, kde je vykonávána reakce na příjem události.

Registrování a odregistrování posluchače

Při obou procesech se postupuje podobným způsobem. Obojí je vykonáváno voláním metody vysílače. Hlavičky metod jsou téměř shodné a jejich parametry mají stejný význam:

```
public void addNotificationListener(NotificationListener listener,
                                    NotificationFilter filter,
                                    Object handback);
public void removeNotificationListener(NotificationListener listener);
public void removeNotificationListener(NotificationListener listener,
                                       NotificationFilter filter,
                                       Object handback);
```

U registrace je předán samotný posluchač. Nepovinnou možností je udání filtrovacího objektu, přes který jsou filtrovány všechny zprávy pro daného posluchače ještě před odesláním. Volitelný objekt označený `handback` slouží pro příjemce k určení kontextu. Vysílající

mbeana ho nesmí modifikovat. Voláním metody `addNotificationListener` dává posluchač najevo připravenost k příjmu událostí.

Odregistrování posluchače se provádí v momentě, kdy nestojíme nadále o příjem událostí nebo kdy rušíme objekt posluchače. Metoda `removeNotificationListener` odstraňuje posluchače ze seznamu. Při mnohonásobné registraci jednoho posluchače (například u spojení s více handback objekty), je třeba mít na vědomí, že u `NotificationBroadcaster` jsou vždy odebrány všechny záznamy posluchače. Proto je v rámci `NotificationEmitter` metoda přetížena druhou se specifikací filtru a handback objektu.

6.1.6 Identifikace mbeany

Jak již bylo zmíněno dříve, jednotlivé instance mbean se registrují ke konkrétním MBean serverům agentní vrstvy. Každý ze serverů obsahuje seznam vedených mbean. K tomu, aby bylo možné odlišit jednu od druhé, je třeba mbeany určitým způsobem identifikovat.

Za účelem jednoznačné identifikace byla definována třída `ObjectName`. Objekty této třídy hrají velmi významnou roli v celém řídícím systému. Vzhledem k faktu, že komunikace mezi řídící aplikací a jednotlivými mbeanami probíhá nepřímo přes MBean server, veškerá volání operací, modifikace atributů i práce s notifikačním modelem jsou doprovázena identifikací konkrétní mbeany formou objektu třídy `ObjectName`.

`ObjectName` můžeme definovat jako textový řetězec s následující syntaxí:

```
domena : klic1 = hodnota1, klic2 = hodnota2, ...
```

Vždy je řetězec sestaven ze dvou logických částí oddělených dvojtečkou:

- *Doména* - jednoznačné označení seskupení mbean. Jedná se o obdobu balíků Javy, jmenných prostor platformy .NET nebo Namespace tagu značkovacího jazyka XML. Zabraňuje kolizím identifikátorů se shodným seznamem atributů. Každý MBean server obsahuje jednu implicitní doménu. Uvedením prázdného řetězce namísto doménového jména označuje náležitost mbeany právě do domény implicitní.

  Každý tvůrce mbean by měl mít svoji vlastní nezaměnitelnou doménu. Například server GlassFish používá doménu *amx* pro všechny své mbeany.

- *Klíčové atributy* - seznam dvojic `klic=hodnota` oddělených čárkou. Mimo identifikace mbeany je možné jimi nastavit počáteční hodnoty vybraných atributů. Tento seznam povinně obsahuje minimálně jedno spojení atributu a hodnoty.

Pro správnou identifikaci mbeany webové služby je vhodné využít vlastní domény a k ní připojit minimálně jeden atribut s unikátní hodnotou. Tento atribut by měl webovou službu plně charakterizovat.

U procesu konstrukce objektu z připraveného řetězce je jeho správná syntaxe kontrolována. Při jakékoliv neshodě s předchozími pravidly je vyvolána vyjímka `MalformedObjectNameException`.

6.1.7 Životní cyklus mbeany

Vytvoření mbeany

Životní cyklus každé mbeany začíná v momentě, kdy vznikne objekt třídy, jež ji implementuje. Vždy ale záleží na kontextu, kde je mbeana umístěna. Objektová reprezentace MBean

serveru - třída `MBeanServer` nám poskytuje možnost instanciace mbeany přetíženou metodou `instantiate`. Zavoláním `instantiate` server vytvoří nový objekt třídy mbeany podle názvu ve vstupním parametru. Samozřejmostí je fyzické umístění třídy na serveru.

V případě mbeany webové služby je třeba čas vytvoření sladit s dobou vzniku samotné služby. Pokud chceme, aby byla mbeana dostupná po celou dobu existence webové služby na serveru, nejlepším řešením je instanciace objektu mbeany přímo z konstruktoru služby.

Registrování mbeany

Proces registrace mbeany je pevně spjat s vybraným MBean serverem. Třída `MBeanServer` disponuje metodou `registerMBean`, jež pomocí objektu mbeany a její identifikace `ObjectName` předaných vstupními parametry zaregistruje mbeanu na serveru. Podmínkou je unikátnost identifikátoru mbeany v rámci serveru. Při jejím nesplnění je volána vyjímka `InstanceAlreadyExistsException` a proces končí neúspěchem.

Třída `MBeanServer` nabízí možnost sloučení procesů instanciace a registrace do jednoho ve formě volání metody `createMBean`. Metoda slouží pro zjednodušení práce, neprovádí nic navíc.

Otázka registrace mbeany webové služby je řešena ve stejném okamžiku jako její instanciace. Tedy pokud máme v úmyslu spravovat webovou službu po celou dobu jejího setrvání na serveru, musíme mbeanu zaregistrovat ihned po vytvoření služby. Ideálním místem je konstruktor služby. Alternativou se může stát samostatná metoda webové služby. Zde je podmínkou doplnění anotace @PostCreate. Takto anotovaná metoda je volána ihned po vytvoření instance služby.

Odregistrování mbeany

Odregistrování je potřeba provést po skončení práce s mbeanou. Provedením odregistrace objekt třídy mbeany stále existuje, není ale přístupný managementu. Opět je pro tento účel definována metoda `unregisterMBean` objektu `MBeanServer`.

Vzhledem k neexistenci „pravého“ destruktoru v jazyce Java autor textu dlouhou dobu řešil, jak zajistit odstranění mbeany webové služby z MBean serveru zároveň se zrušením objektu Webové služby. Na garbage collector se v tomto případě nelze úplně spolehnout. Nakonec se ukázalo nejjednodušším řešením odregistrace mbeany služby v jakékoliv metodě s anotací `@Predestroy`. S tímto přístupem je mbeana úspěšně odregistrována vždy s koncem webové služby a nezůstává v̈isetïa serveru.

Závislost na životním cyklu webové služby

Z hlediska životního cyklu se webová služba chová odlišně od standardní Java aplikace. Služba může být implementována formou Enterprise Java Bean (EJB) prvku uvnitř EJB modulu nebo, jako ve většině případů, formou Servletu umístěného v rámci webového modulu. Instanciaci i rušení služby řídí vlastnický kontejner - podle implementace EJB nebo webový kontejner. Tvůrce služby na tento proces nemá vliv.

Webovou službu umisťujeme ve formě WAR nebo EAR archivů. Při jejich instalaci se, mimo fyzického transferu dat, připraví server na použití služby. To zahrnuje i vytváření a registraci serverem připravených mbean.

Pokud jsou na serveru připraveny generické mbeany pro kontrolu a monitorování webových služeb, mohou vzniknout současně s instalací webové služby na server. Jejich registrace je provedena ještě před vytvořením samotné instance webové služby. V situaci, kdy je

webová služba dodatečně instruována prostředky pro management, nejsme schopni provést registraci mbeany dřív, než při vytvoření instance třídy služby.

Aplikace pro management a monitorování webových služeb je postavena na datech z jednotlivých mbean. Zatímco informace z většiny generických mbean jsou dostupné po celou dobu setrvání služby na serveru, data speciálně instruovaných mbean můžeme využít až po vytvoření objektu webové služby, tzn. po prvním volání služby.

6.2 Připojení k MBean serveru

Jak již bylo zmíněno, komunikace mezi řídící aplikací a jednotlivými mbeanami neprobíhá přímo, ale přes JMX agenta. Prvním krokem řídící aplikace pracující s mbeanami webových služeb je ustanovení spojení s MBean serverem. Zde předpokládáme, že řídící aplikace je lokalizována v jiném virtuálním stroji než spravované webové služby a používá tak vzdáleného přístupu k MBean serveru. U komunikace v rámci jednoho virtuálního stroje nejsou zapojeny komunikační protokoly a tudíž není třeba konstrukce spojení.

6.2.1 JMXServiceURL

Ke spojení aplikace s MBean serverem je využíváno konektorů. Podle popisu v sekci 5.2.3 víme, že je konektor rozdělen na dvě části - serverovou a klientskou. Pro ustavení spojení potřebujeme znát umístění serverové části konektoru našeho MBean serveru. Umístění je specifikováno ve formě `JMXServiceURL`, která má tvar:

```
service:jmx:<protokol>:<umístění konektoru>
```

Praktický příklad vypadá následovně:

```
service:jmx:rmi:///jndi/rmi://skritek:8686/jmxrmi
```

Tento zápis adresy vypovídá o použití implicitního konektoru RMI. Hledá se v RMI registrech na adrese `skritek:8686`. Poslední část `jmxrmi` je unikátním označením běžící JMX služby na zadané adrese.

6.2.2 Standardní připojení JMX

Připojení k MBean serveru zahrnuje dvě související fáze. První z nich je vytvoření klientské části konektoru ke zkompletování konektoru. Druhou fází je získání proxy objektu MBean serveru, přes který jsou ovládány všechny vzdálené mbeany.

Vytvoření klientské části konektoru

Adresa serverové části konektoru ve formátu `JMXServiceURL` je vždy vystavena vzdáleným serverem. Její klientský protějšek vznikne ustavením připojení. Pro usnadnění tohoto procesu je definována přetížená statická metoda `connect` třídy `JMXConnectorFactory`:

```
static JMXConnector connect(JMXServiceURL serviceURL);
static JMXConnector connect(JMXServiceURL serviceURL,
                            Map<String,?> environment);
```

Jako parametry uvedeme adresu serverové části konektoru a dvojici správného přístupového jména a hesla. Vrácený objekt třídy `JMXConnector` reprezentuje konektor na straně řídící aplikace.

Získání proxy objektu MBean serveru

Stejně jako při lokálním volání využíváme MBean serveru v podobě objektu implementujícího rozhraní `MBeanServer`, i v případě vzdáleného přístupu potřebujeme nějakou reprezentaci MBean serveru. Stává se jí proxy objekt rozhraní `MBeanServerConnection`. Získáme jej jednoduchým voláním operace `getMBeanServerConnection` nad objektem konektoru.

Rozhraní `MBeanServerConnection` je předkem `MBeanServer` a reprezentuje MBean server nezávisle na lokalitě přístupu. Oproti lokální verzi `MBeanServer` neumožňuje instanciaci a registraci mbean metodami `instantiate` a `createMBean` zmíněných v sekci 6.1.7.

6.2.3 Připojení k serveru GlassFish

Pokud máme v plánu využívat specifických služeb GlassFish serveru, musíme zvolit odlišný způsob připojení. GlassFish definuje svou vlastní třídu `AppserverConnectionSource` pro udržování informací o spojení se serverem. Instanciaci objektu této třídy a tím i připojení k MBean serveru lze vytvořit opět pomocí jedné ze statických metod třídy `Connect`:

```
static AppserverConnectionSource
       connectNoTLS(String host, int port,
                    String user, String userPassword)

static AppserverConnectionSource
       connectTLS(String host, int port,
                  String user, String userPassword,
                  boolean promptForUnknownCertificate)
```

Výběr metody závisí na aplikovaném způsobu připojení. Metoda `connectTLS` formuje zabezpečené spojení Transport Layer Security (TLS), `connectNoTLS` nikoliv. Přes objekt třídy `AppserverConnectionSource` získáme výchozí přístupový bod ke struktuře mbean specifických GlassFish serveru stejně tak jako proxy objekt `MBeanServerConnection` pro práci s mbeanami ostatními.

6.3 Způsoby manipulace s mbeanami

V okamžiku, kdy bylo úspěšně navázáno spojení mezi řídící aplikací a MBean serverem lze začít přistupovat k mbeanám. Technologie Java Management Extensions definuje dva různé přístupy v této oblasti. Prvním z nich je nepřímá manipulace s mbeanami. Druhým způsobem je zprostředováni interakce lokálně umístěným proxy objektem mbeany.

6.3.1 Nepřímá manipulace

Ve fázi připojení jsme získali proxy objekt MBean serveru formou instance rozhraní `MBeanServerConnection`. Při tomto způsobu manipulace s mbeanami jej využíváme k delegaci veškerých volání vybrané mbeaně. Proto je tento přístup označován za nepřímou manipulaci.

Metody `MBeanServerConnection` pro interakci s mbeanami jsou totožné s metodami rozhraní `DynamicMBean` znázorněného na obrázku 6.2. Jediným rozdílem oproti `DynamicMBean` je přídání parametru třídy `ObjectName` pro jednoznačnou identifikaci volané mbeany a množství případně vyvolaných vyjímek k ošetření.

Podmínku pro správnou aplikaci volání metod i manipulaci s atributy je znalost metadat vybrané mbeany. Objekt metadat `MBeanInfo` lze obdržet voláním metody `getMBeanInfo` objektu mbean serveru.

Atributy

Manipulace s atributy zahrnuje získávání a nastavován aktuálních jejich hodnot. `MBeanServerConnection` definuje dva páry metod:

- `getAttribute, getAttributes` - získání hodnoty jednoho či více atributů. Všechny hodnoty jsou navráceny v podobě datového typu definovaného v metadatech mbeany. Při volání vícenásobného getteru je výsledek ve formě seznamu objektů třídy `Attribute`, což není nic jiného než pojmenovaná hodnota.
- `setAttribute, setAttributes` - nastavování hodnot jednoho nebo více atributů. Opět se počítá s užitím instancí `Attribute` pro identifikaci dvojic *atribut, hodnota*.

Operace

Pro volání operací mbean je esenciální metoda `invoke`. Její hlavička vypadá následovně:

```
Object invoke(ObjectName name, String operationName,
              Object[] params, String[] signature)
              throws InstanceNotFoundException, MBeanException,
                     ReflectionException, IOException
```

Spolu s identifikací mbeany předáváme název operace, pole vstupních parametrů a signaturu operace. Signaturu definujeme jako pole textových řetězců odpovídajících kompletním názvům tříd datových typů pole vstupních parametrů. Po získání výsledku je potřeba jej přetypovat na správný datový typ podle metadat.

Notifikace

Zároveň s delegací volání operací a práce s atributy slouží MBean server jako delegát zasílaných notifikací ze strany mbean. Rozhraní `MBeanServerConnection` proto poskytuje nástroje pro registraci a odregistraci poslechu událostí ze strany mbeany. Jde o metody `addNotificationListener` a `removeNotificationListener` popsané v 6.1.5. Pro specifikování mbeany je opět připojen parametr třídy `ObjectName`.

Ošetření vyjímek

Podmínkou k využití nepřímé manipulace je ošetření všech možných výskytů vyjímek při volání metod MBean serveru. Jejich nemalý počet je sestaven z vyjímek specifických každé implementaci mbeany a z vyjímek spojených s manipulací mbean obecně. Za nejčastější vyjímky se dají považovat:

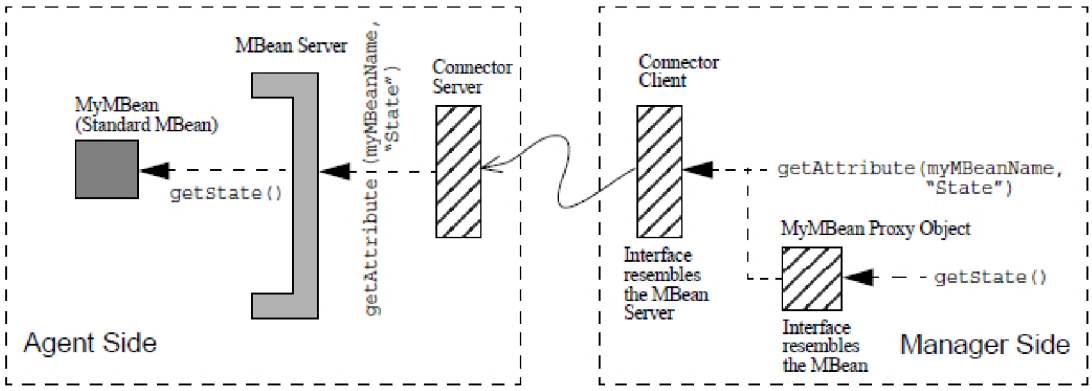
- `InstanceNotFoundException` - Mbean server nenalezl mbeanu podle identifikace `ObjectName`. Nejspíše hledaná mbeana není na serveru registrována nebo jsme uvedli její nekorektní název.
- `MBeanException` - Zaobalení vyjímek definovaných tvůrcem mbeany. Ve většině případů jde o nesprávný rozsah hodnot parametrů. Původní vyjímku získáme voláním metod `getTargetException` nebo `getCause` nad objektem vyjímky obalující.

- `ReflectionException` - velmi častá vyjímka. Zaobaluje v sobě vyjímky spojené s nesprávným voláním metod MBean serveru. Setkáme se sní v případě zadání špatné signatury operace nebo konstruktoru, nesprávného názvu atributu a v podobných situacích. Nejde jen o případy zkomolených názvů, datových typů, ale i momenty, kdy neoznačíme metody rozhraní za veřejné klíčovým slovem `public`. Stejně jako u `MBeanException` se k iniciační vyjímce dostaneme voláním `getTargetException` nebo `getCause`.
- `RuntimeOperationsException` - vyjímka vznikající při výskytu prázdné hodnoty `null` u parametrů očekávajících nenulovou hodnotu.
- `IOException` - pochybení v komunikaci mezi řídící aplikací a MBean serverem. Setkáváme se s ním i při práci s proxy objekty mbean.

Vyjímky jsou nejen u managementu silným reflexním mechanismem. U nepřímé manipulace jich vzniká mnoho. Porozumění významu všech vyjímek pomůže při řešení způsobu jejich ošetřování.

6.3.2 Přístup přes proxy

Druhou cestou, jak komunikovat s mbeanami, jsou proxy objekty. V rámci JMX se dá proxy definovat jako lokálně umístěný objekt implementující rozhraní mbeany, kterou na straně řídící aplikace zastupuje. Proxy mbean konstruujeme žádostí MBean serveru. Voláním metod proxy jsme schopni ovládat mbeanu aniž bychom znali její přesné umístění.



Obrázek 6.3: Definice rozhraní DynamicMBean (převzato z [8])

Ve skutečnosti se princip komunikace mezi MBean serverem a klientskou aplikací neliší. Zatímco u nepřímé manipulaci musí vše obstarat programátor, zde je logika schována za implementaci proxy objektu. Práce s atributy, operacemi ani s událostmi je shodná u mbeany i u jejího proxy zastoupení.

Obrázek 6.3 zobrazuje volání mbeany přes její proxy. Posloupnost kroků můžeme popsat následovně:

1. Řídící aplikace zavolá nad mbeanou `MyMBean` metodu `getState` pro vyhodnocení atributu.

2. V proxy objektu se volání transformuje do stavu přijatelného MBean serverem `getAttribute(MyMBeanName, "State")`. Parametr `MyMBeanName` znamená identifikace mbeany instancí `ObjectName`.

3. Následuje přenos volání na server, kde je podle názvu mbeany zrekonstruováno volání zpět do původního stavu.

4. Vrácený výsledek volání se od mbeany dostane stejnou cestou zpět k jejímu proxy zástupci.

Konstrukci proxy objektu zajišťujeme s pomocí statické knihovny `JMX` standardního balíku `javax.management` a jejích přetížené metody:

```
static <T> T newMBeanProxy(MBeanServerConnection connection,
                           ObjectName objectName,
                           Class<T> interfaceClass);
static <T> T newMBeanProxy(MBeanServerConnection connection,
                           ObjectName objectName,
                           Class<T> interfaceClass,
                           boolean notificationBroadcaster);
```

Povinnými vstupními parametry jsou objekty reprezentující Mbean server a mbeanu cílové proxy (`connection, objectName`). Jsme nuceni zadat i třídu pro mapování proxy `interfaceClass` ve formátu instance `java.lang.Class`. Volitelný je příznak signalizace `notificationBroadcaster`, zda má proxy objekt šířit notifikace. Posledně zmíněného využíváme, pokud máme v plánu registrovat posluchače události tohoto prvku.

Zde je konkrétní vzorek kódu vytvářející mbeanu webové služby:

```
try{
  ...
  ObjectName mbeanObjectName = new ObjectName('me.services:type=HelloWS');
  HelloWSMBean serviceProxy = JMX.newMBeanProxy(serverConnection,
                                                mbeanObjectName,
                                                HelloWSMBean.class);
  ...
}
```

6.3.3 Srovnání nepřímé manipulace a proxy objektů

Při aplikování obou způsobů komunikace s mbeanami docházíme k závěru, že každý z nich nese své výhody i nevýhody. Ve většině situací záleží na tvůrci řídícího systému, jakou cestu zvolí. Někdy je ovšem možnost výběru omezena.

Kladné vlastnosti přístupu přes proxy mbean se dají specifikovat jako:

- *Nenáročnost použití* - Proxy je normální objekt a tak se s ním zachází. Programátor nepotřebuje znát detaily spojení, strukturu metadat mbeany ani specifické znaky MBean serveru. Veškerá nestandardní logika je ukryta v implementaci proxy.

- *Nízká režie kódu* - Objem napsaného kódu klesá. Současně se zvyšuje přehlednost.

- *Předdefinované proxy* - Některé servery nabízejí své vlastní proxy objekty. Například JMX systémy serveru GlassFish jsou postaveny na proxy objektech. Více v kapitole 7.

Naopak výhody nepřímé manipulace s mbeanami se dají charakterizovat jako:

- *Obecná použitelnost* - Existují situace, kdy nelze proxy objekty použít. Příkladem jsou generické řídící aplikace. Zde dopředu neznáme, jaké mbeany budeme spravovat. Nevlastníme jejich management rozhraní, z něhož bychom mohli proxy objekt vytvořit.
- *Předvídatelnost chování* - Skrytím implementace za proxy objekt ztrácíme možnost jakkoliv ovlivnit tuto oblast. Nejsme schopni se vyhnout chybám a nedostatkům implementace. U nepřímé manipulace sice napíšeme velká kvanta zdrojového kódu, ale přesně víme, jak se bude aplikace chovat.

Ideálním způsobem práce s mbeanami se zdá být kombinace obou přístupů. Jako hlavní prostředek se jeví aplikace proxy objektů. V místech, kde nemůžeme zkonstruovat proxy, provedeme implementaci nepřímou manipulací.

Kapitola 7

Aplikační rozhraní AMX serveru GlassFish

7.1 AMX mbeany

GlassFish patří mezi skupinu serverů, které disponují silně rozvinutou podporou managementu. Pro přípravu řídících systémů představuje aplikační rozhraní nazvané Application Server Management Extensions (AMX). Vše je postaveno na bázi Java Management Extensions ovšem na mnoha místech podstatně rozvinuto. Tato kapitola popisuje rozhraní AMX včetně vylepšení oproti standardnímu JMX. Při studiu rozhraní AMX jsme používali dokumentaci ([9]) a průvodce poskytnutého výrobcem aplikačního serveru ([11]).

Kompletní rozhraní AMX je umístěno v balíku `com.sun.appserv.management`. Vystavuje veškeré mbeany serveru ve formě proxy objektů implementující jejich řídící rozhraní. I při použití proxy stále zachovává možnost nepřímé manipulace mbean přes MBean server. Nutno poznamenat, že předdefinovanými proxy objekty komunikuje pouze s vlastními mbeanami zasazenými do domény `amx`. Spravované entity definované uživatelem z výhod AMX proxy těžit nemohou.

Pro ujasnění terminologie je potřeba zmínit, že při popisování aplikačního rozhraní AMX budeme pod termínem AMX mbeana rozumět samotný proxy objekt na straně klienta.

7.1.1 Specifičnost AMX mbean

Všechny mbeany serveru fungují dle standardu JMX systémů. Pro usnadnění jejich identifikace, rozdělení do kategorií či vzájemná navigace jsou mbeany specifické serveru (běžné je jejich označení předponou AMX) doplněny o povinnou implementaci základního rozhraní AMX.

Následující výčet uvádí ty nejpodstatnější atributy každé AMX mbeany vynucené implementací zmíněného rozhraní:

- `name` - Samotný název mbeany. Užitečný při vyhledávání souvisejících prvků na serveru - například při identifikace mbean vybrané webové služby.
- `j2eeType` - Typ prvku reprezentovaného mbeanou. Jako příklad lze uvést webový modul, virtuální stroj Javy, servlet ap.
- `group` - příznak náležitosti do specifické skupiny (princip rozdělení do skupin je popsán v následujícím textu).

- `domainRoot` - odkaz na objekt mbeany sloužící jako výchozí bod celého AMX.
- `container` - odkaz na vlastníka mbeany. Prostředek pro definování struktury vztahů.

AMX rozhraní je potomkem `NotifiacationBroadcater`, tudíž všechny AMX mbeany plní implicitně roli vysílače událostí.

Na serveru se vyskytují mbeany rozličných zaměření představující různou funkčnost. K tomu, abychom mezi nimi mohli jednoduše vyhledávat a procházet, jsou mbeany serveru GlassFish dodatečně rozdělovány do připravených skupin právě podle jejich účelu. Proto každá z mbean obsahuje parametr s názvem *group*, jenž lze jednoduše získat voláním metody `getGroup`.

Skupiny AMX mbean jsou:

- *konfigurační* - tento typ mbean slouží k nastavování spravovaného prvku. Často bývá vázán na konfigurační soubory formátu XML umístěné na serveru.
- *monitorovací* - určeny ke sledování parametrů spravované entity. Množství, datové typy i granularita monitorovaných informací se odvíjí od typu sledovaného prvku.
- *JSR77* - mbeany sestrojené na základě J2EE Management modelu (viz 5.3). Podle tohoto modelu reprezentují spravovatelné prvky J2EE serverů. Specifikace JSR prosazuje užívání metod `getMonitoringPeer` a `getConfigPeer` k získávání odpovídajících monitorovacích a konfiguračních mbean. Metody však ne vždy plní svou funkci na sto procent.
- *užitné (z angl. utility)* - prvky vykonávající obecně dostupné služby serveru.
- *ostatní* - mbeany, které nemůžeme zařadit ani do jedné z předešlých kategorií.

7.1.2 Navigace mezi AMX mbeanami

Při práci s mbeanami v řídící aplikaci je jednou z nejčastějších otázek, které programátor řeší, způsob získávání požadovaných mbean. GlassFish server ve své druhé verzi jich ihned po instalaci obsahuje řádově desítky, spíše stovky. Se zvyšujícím se počtem spravovaných prvků narůstá i množství instruovaných mbean. Proto vzniká potřeba způsobu, který je dovolí mbeany rychle a efektivně vyhledávat a navigovat mezi nimi. GlassFish a jeho proxy objekty nabízejí tři možnosti - cestu kontejnerů, dotazovacího jazyka nebo přímého procházení.

Přímé procházení

Princip přímého procházení mezi AMX mbeanami souvisí s jejich vzájemnými vztahy a sdružováním podle kategorií. Pro práci s různými skupinami mbean definuje GlassFish server své vlastní prostředky. Jsou jimi opět mbeany někdy doplňovány přídomkem „Mgr“ jako signalizace manažerů. Jako příklad lze uvést mbeany pro oblast monitorování (`JMXMonitorMgr`), konfigurace (`DomainConfig`) nebo webových služeb (`WebServiceMgr`).

Při tomto postupu se využívá skutečnosti, že jsou mbeany vzájemně provázány a tvoří určitou hierarchii. Jejím vrcholkem a výchozím bodem je objekt třídy `DomainRoot`, jehož primárním účelem je sloužit jako kořen navigace. Získáme jej přímo z MBean serveru. Poté lze využít metod getterů pro získání manažerů a přes něj přistupovat dále k hledaným entitám. Každá AMX mbeana zahrnuje metodu `getDomainRoot` pro bezproblémový návrat.

Následující ukázka kódu znázorňuje, jak při použití přímého procházení získáme seznam všech koncových bodů webových služeb na serveru:

```
...
DomainRoot domainRoot = mbeanServerConnection.getDomainRoot();
WebServiceMgr wsManager = domainRoot.getWebServiceMgr();
Map<Object, String> keyMap = wsManager.getWebServiceEndpointKeys();
for(Object key: keyMap.getKeySet) {
  WebServiceEndpoint endpoint
    = wsManager.getWebServiceEndpointSet(key, WebServiceMgr.ALL_SERVERS);
...
```

Přímé procházení je nejjednodušším cestou, jak najít hledaný cíl. Najdou se však i situace, kdy je procházení hierarchií natolik zdlouhavé, že se vyplatí využít ostatních řešení.

Kontejnery

Druhou cestou je práce s kontejnery. Opět je vše postaveno na stromové hierarchii. Konkrétněji se zde využívá také faktu, že každá mbeana vystupující jako vlastník ostatních implementuje rozhraní `Container`. Toto rozhraní poskytuje modifikace metod pro vyhledávání vlastněných mbean. Můžeme vyhledávat podle názvů i typů mbean. Na druhou stranu jsme schopni procházet hierarchií směrem ke kořenu voláním metody `getContainer` povinně obsažené v každé AMX mbeaně.

Výpis zdrojového kódu představuje velmi zjednodušenou posloupnost kroků pro získání všech webových modulů serveru „server“ domény „domain1“:

```
...
DomainRoot domainRoot = mbeanServerConnection.getDomainRoot();
Container container = (Container)domainRoot;
J2EEDomain domain = (J2EEDomain)containergetConatinee(
                          J2EETypes.J2EE_DOMAIN, 'domain1');
J2EEServer server = (J2EEServer)containergetConatinee(
                          J2EETypes.J2EE_SERVER, 'server');
Map<String, ?> wmMap =  domain.getContaineeMap(J2EETypes.WEB_MODULE);
for(String key: wmMap.keySet()){
  WebModule module = (WebModule)wmMap.get(key);
...
```

Již podle těchto řádků se dá jednoduše odvodit, že se nejedná zrovna o nejrychlejší a nejpraktičtější postup. Kroky v posloupnosti se neustále opakují. Na každé úrovni hierarchického stromu získáme jeden nebo více prvků, provedeme přetypování a využijeme jej k průchodu na úroveň nižší.

Stejně jako u přímého procházení i zde prostupujeme stromem hierarchie. Nevýhodou je narůstající složitost s hlubším zanořením. Navíc se zdrojový kód prodlouží o operace průchodu mapami a ošetřování správnosti návratových hodnot.

Dotazovací vyhledávání

Poslední cestou, jak operovat s mbeanami, je využití mechanismu hledání mbean s pomocí speciálně formulovaných dotazů. Narozdíl od předchozích dvou způsobů se zde pohlíží na

obsah MBean serveru jako na jednu velkou kolekci prvků, které nejsou mezi sebou nijak propojeny. Vše je založeno na procházení na shodě hledaného vzoru s textovým identifikátorem mbeany `ObjectName`.

Samotné JMX má v sobě zakomponovánu možnost hledání pomocí definovaných výrazů. GlassFish server na JMX základu staví a připravuje proxy objekt zvaný `QueryMgr`. Ten obstarává logiku konstrukce dotazů na základě názvů, typů i domén hledaných entit.

V následujícím příkladě hledáme opět všechny koncové body webových služeb na serveru:

```
...
DomainRoot domainRoot = mbeanServerConnection.getDomainRoot();
QueryMgr manager = domainRoot.getQueryMgr();
Set s~= manager.queryJ2EETypeSet(J2EETypes.WEB_SERVICE_ENDPOINT);
Iterator it = s.iterator();
while(it.hasNext()) {
  WebServiceEndpoint endpoint = (WebServiceEndpoint)it.next();
...
```

Dotazovací vyhledávání nahlíží na všechny mbeany stejným způsobem nezávisle na jejich pozici v hierarchické stromové struktuře. Nezáleží tedy, zda pracujeme s kořenovým singletonem[1] `DomainRoot` nebo s některou mbeanou na nižších úrovních. Navíc oproti ostatním přístupům můžeme užít tohoto hledání i v případě mbean mimo doménu `amx` serveru GlassFish.

7.2 Generické mbeany webových služeb

Při sestavování aplikace pro management webových služeb počítáme s faktem, že budeme využívat všech prostředků a pracovat se službami pro management speciálně upravenými. Jinými slovy bereme za samozřejmost práci s řídícím rozhraním mbean webových služeb. Díky rozhraním máme k monitorování i managementu zpřístupněny operace i atributy specifické každé webové službě.

Existují však i další informace o webových, které můžeme sledovat. Stará se o ně přímo samotný server a jejich přítomnost není podmíněna zvláštní instrumentací. Obecně se jedná o data týkající se implementace služeb, jejich umístění a provozu na serveru. GlassFish server užívá ke zveřejnění těchto údajů čtyř typů proxy objektů - `WebServiceEndpointInfo`, `WebServiceEndpoint`, `WebServiceEndpointConfig` a `WebServiceEndpointMonitor`.

WebServiceEndpointInfo

Proxy objekt třídy `WebServiceEndpointInfo`, jak lze jednoduše podle názvu odvodit, obsahuje základní informace o webové službě. Server jej vytváří ihned po instalaci služby. Vyskytuje se vždy v jediné instanci a zpřístupňuje data spojená hlavně s implementací. `WebServiceEndpointInfo` je složen z atributů pouze pro čtení a nedisponuje žádnými operacemi.

Souhrn informací, které můžeme z proxy objektu získat:

- *Identifikace a umístění* - spolu s názvem služby máme přístup i k modulu v jehož rámci je uložena. Navíc lze ověřit, jestli je modul samostatný nebo funguje jako součást podnikové aplikace.

[1] Termínem singleton máme na mysli třídu s jedinou možnou instancí

- *Implementace* - můžeme zjistit, jakým způsobem je služba implementována (EJB nebo Servlet), název a umístění zdrojového souboru.
- *Zabezpečení* - přítomna je indikace zabezpečení webové služby.
- *WSDL* - metadata webové služby ve formě WSDL souboru jsou zprostředkována jediným atributem.
- *Soubory deskriptorů* - součástí jsou i textové atributy držící obsah jednotlivých deskriptorů *webservices.xml*, *web.xml*, *application.xml* a ostatních.

Proxy objekt `WebServiceEndpointInfo` je užitečný hlavně při prvotní identifikaci webové služby. Poskytuje množství statických informací pouze pro čtení, není proto zajímavý z pohledu monitorování. Zajímavé je taky zprostředkování obsahu konfiguračních souborů.

Záhadou je dostupnost popisu webové služby souborem WSDL. Autorovi tohoto textu se bohužel nepodařilo přijít na to, proč je atribut pro WSDL soubor vždy prázdný. Děje se tak při více různých způsobech instalace služby a nezávisle na faktu, že fyzicky je soubor WSDL na serveru přítomen.

WebServiceEndpoint

AMX mbeany typu `WebServiceEndpoint` fungují jako reprezentace koncových bodů webových služeb. Samotný koncový bod může být provozován na více virtuálních serverech. Proto je počet instancí `WebServiceEndpoint` odvozen od počtu serverů, na kterých je koncový bod aplikován.

Tento proxy objekt je významný z pohledu monitorování provozu mezi webovou službou a jejími klienty. Udržuje seznam přijatých i odeslaných zpráv pro zastoupený koncový bod formou statického pole. Zprávy jsou zastoupeny objektem třídy `MessageTrace` a dovolují nám získat data tvořící:

- *Odesílatel* - IP adresa zdroje, název aplikace a uživatele
- *Přijaté zprávy* - HTTP hlavičky přijatých zpráv, kompletní XML obsah SOAP zprávy, její celková velikost
- *Odeslané zprávy* - shodné s přijatými - HTTP hlavičky odeslaných zpráv, XML obsah SOAP zprávy, objem dat.
- *Chyby* - u neúspěšných volání máme k dispozici zdroj chyby, její typ i obsah chybové zprávy.
- *Časové údaje* - čas přijetí, doba potřebná k odeslání

Spolu se vystavením zpráv lze touto AMX mbeanou resetovat celkové monitorovací statistiky koncového bodu.

Proxy objekt `WebServiceEndpoint` vychází ze specifikace managementu JSR77. Je součástí řídícího modelu GlassFish serveru a představuje spravovaný objekt `J2EEManagedObject` (více v sekci 5.3). Jeho povinností je proto implementace metod `getMonitoringPeer` a `getConfigPeer` pro zpřístupnění monitorovacích a konfiguračních prvků. Zjednodušeně řečeno voláním zmíněných dvou metod nad koncovým bodem jsme schopni se dostat k prostředkům pro konfiguraci a monitorování koncového bodu nebo celé služby.

Při testování obou metod však vychází najevo, že jejich volání ne vždy funguje podle předpokladů. U mnohých koncových bodů je metodami vrácena prázdná hodnota jako signalizace nenalezení výsledku. Přitom hledáním jinou cestou nás ubezpečí, že monitorovací a konfigurační mbeany jsou vytvořeny a úspěšně registrovány na serveru. Z tohoto důvodu autor tohoto textu jejich aplikaci nedoporučuje.

WebServiceEndpointConfig

AMX mbeana `WebServiceEndpointConfig` představuje právě konfigurační nástroj webové služby. Stejně jako u `WebServiceEndpointInfo` vždy existuje jen jediná instance pro celou službu nezávisle na počtu virtuálních serverů. K této mbeaně přistupujeme v okamžiku, kdy chceme změnit nastavení její webové služby.

Konfigurační proxy nastavuje formou atributů umístění webové služby, transformační pravidla formátu XSLT pro příchozí i odchozí zprávy. S pomocí této proxy nastavujeme úroveň monitorování webové služby atributem `MonitoringLevel`. Jeho rozsah je omezen na pouhé tři hodnoty:

- *OFF* - hodnota indikuje kompletně vypnutý monitoring celé služby. Nesledují se statistiky provozu ani přenášené zprávy. Jde o implicitní hodnotu uvedenou po instalaci každé služby.
- *LOW* - nízká úroveň se soustředí na sledování pouze statistik provozu. Nastavujeme ji tehdy, když nás nezajímá kdo a jakým způsobem komunikuje se službou.
- *HIGH* - nastavením nejvyšší hodnoty *HIGH* dáváme najevo zájem o zaznamenání všeho, co se sledovat dá. Počet zpráv, které se drží v historii, je omezen hodnotou atributu `MaxHistorySize` stejné mbeany.

U této AMX mbeany je potřeba zmínit odlišnost času jejího vytvoření a registrace od předchozích dvou typů. GlassFish server iniciuje vytvoření konfigurace webové služby až na základě jejího prvního použití. Dokud není služba zavolána, instance `WebServiceEndpointConfig` neexistuje. Pokud potřebujeme inicializovat konfiguraci před prvním zavoláním lze tak učinit získání vlastnického modulu služby a vytvořením `WebServiceEndpointConfig` mbeany metodou modulu.

WebServiceEndpointMonitor

Poslední generickou mbeanou GlassFish serveru pro práci s webovými službami je `WebServiceEndpointMonitor`. Stejně jako reprezentace koncových bodů `WebServiceEndpoint` se jejich množství náležící jedné službě liší podle počtu virtuálních serverů.

Hlavním úkolem tohoto proxy objektu je sledovat zatížení jednotlivých koncových bodů služby a počítat statistické údaje. Ty jsou zveřejněna formou objektu třídy `WebServiceEndpointAggregateStats`, který drží následující položky:

- *Čas všech odpovědi* - minimální, průměrná a maximální doba strávené mezi příchodem požadavku a odesláním odpovědi.
- *Čas poslední odpovědi* - čas strávený posledním voláním
- *Propustnost* - průměrný počet zpracovaných zpráv za minutu
- *Počty zpracování* - čítače úspěšně i neúspěšně zpracovaných žádostí

- *Počty autorizací* - čítače úspěšně i neúspěšně zpracovaných autorizací

Sbírané statistiky jsou pevně spojeny s umístěním koncového bodu na vybraném virtuálním serveru. Voláním metody `resetStats` odpovídajícího koncového bodu `WebService-Endpoint` je nulujeme.

Celkově je přítomnost těchto proxy objektů závislá na nastavení atributu `Monitoring-Level` konfigurační AMX mbeany. Implicitně je po vytvoření konfiguračního prvku vše, včetně úrovně monitorování, vypnuto. Instance monitorovací jednotky vznikne a zaregistruje se po přepnutí na *LOW* nebo *HIGH*. Po opětovném zablokování hodnotou *OFF* mbean nesetrvává, server se postará o její zničení.

Z pohledu řídící aplikace je monitorovací mbeana užitečná pro sledování zátěže a výše chybovosti webové služby na serveru. Sledováním časů průměrných odpovědí a čítače chybových volání jsme schopni detekovat přetížení nebo například úplný výpadek a následně vznikou situaci řešit.

7.3 Monitorování atributů

Doposud jsme v textu probírali oblast mbean, jejich atributů a operací, jež dokážou řídit. To vše se týká správy a řízení zveřejněných prvků. Oblastí velmi úzce spjatou s managementem je monitorování. K tomu, abychom mohli interaktivně řídit systém, potřebujeme kromě samotných nástrojů kontroly také prostředky pro zjišťování stavů kontrolovaných prvků.

Jedním z těchto prostředků, které zajišťují aktivní spolupráci mbean s řídícími systémy jsou notifikace. Jejich princip v rámci notifikačního modelu JMX je popsán v sekci 6.1.5. Zkráceně jde o proces, kdy samotná mbeana iniciuje akci v klientské aplikaci na základě změny svého vnitřního stavu. Nevýhodou tohoto principu je, že pro vysílání událostí musí být mbeana prvku uzpůsobena. Navíc nelze dopředu předpokládat, co budeme chtít u prvku kontrolovat.

Druhou cestou je využití monitorovacích služeb, které jsou povinnou součástí agentní vrstvy JMX systémů (více o agentní vrstvě v sekci 5.2.2). Termín monitorovací služby označuje systém tzv. monitorů. Monitory kombinují výhody notifikačního modelu a skutečnosti, že lze jednoduše přistupovat z jedné mbeany k obsahu druhé. Nejedná se tedy o nic jiného, než o systémem připravenou mbeanu, která sleduje hodnoty atributů mbeany jiné a na základě splnění určité podmínky vysílá notifikaci všem svým posluchačům. Odstraňuje tak jistým způsobem nedostatky výše zmíněné první varianty - nevyžaduje úpravu sledované mbeany a dovoluje nastavit sledování libovolného atributu.

7.3.1 Společný základ monitorování

Pro správu monitorů je na GlassFishi připravena proxy `JMXMonitorMgr`, která plní funkci registru všech monitorů. Monitory vytváříme i rušíme také přes tuto proxy. Jako součást JMX standardu můžeme monitory využívat samozřejmě i mimo aplikační rozhraní AMX.

Celkově jsou v rámci monitorovacích služeb provozovány tři druhy monitorů. Rozdělujeme je podle způsobu sledování atributů na řetězcové, čítací a rozsahové monitory (z angl. string, counter a gauge). Všichni vycházejí ze společného základu implementací rozhraní `MonitorMBean`. Můžeme toho využít popisem funkčnosti, kterou všichni sdílejí.

Monitor je schopný sledovat pouze jediný atribut identifikovaný názvem. Existuje zde však možnost sledovat jej u více mbean současně. Počet posluchačů událostí není omezen. Monitor lze spouštět a zastavovat metodami `start` a `stop`. Sledování a zasílání notifikací

probíhá vždy jen ve spuštěném stavu. Periodicita kontroly je zajištěna využitím další ze služeb JMX agenta - časovače. Hodnoty atributu mbean se porovnávají na základě příchozích událostí od časovače. My jsme schopni nastavit pauzu mezi kontrolami metodou `setGranularityPeriod`.

Obecný způsob použití monitorů můžeme popsat následující posloupností kroků:

1. Vytvoříme a zaregistrujeme mbeanu monitoru.
2. Nastavíme jeho parametry sledování (co, jakým způsobem a jak často budeme sledovat).
3. Zaregistrujeme posluchače událostí monitoru. Můžeme použít filtr událostí.
4. Spustíme sledování.
5. Po zastavení monitorování odregistrujeme příjem událostí a zrušíme mbeanu.

Pro názornou ilustraci vytvoříme v následujícím úryvku kódu řetězcový monitor *mujMonitor*, který sleduje atribut *name* webové služby *MBeanHelloWS*. Porovnává jeho hodnotu s řetězcem „Ondra“ jednou za sekundu a zasílá události při rozdílu i při shodě řetězců. Jako posluchač událostí slouží samotná implementační třída řídící aplikace:

```
...
DomainRoot domainRoot = mbeanServerConnection.getDomainRoot();
JMXMonitorMgr manager = domainRoot.getJMXMonitorMgr();
AMXStringMonitor monitor = manager.createStringMonitor('mujMonitor');
monitor.addObservedObject(new ObjectName('me.services:type=MBeanHelloWS'));
monitor.setObservedAttribute('name');
monitor.setStringToCompare('Ondra');
monitor.setNotifyDiffer(true);
monitor.setNotifyMatch(true);
monitor.setGranularityPeriod(1000);
monitor.addNotificationListener(this, null, null);
monitor.start();
...
```

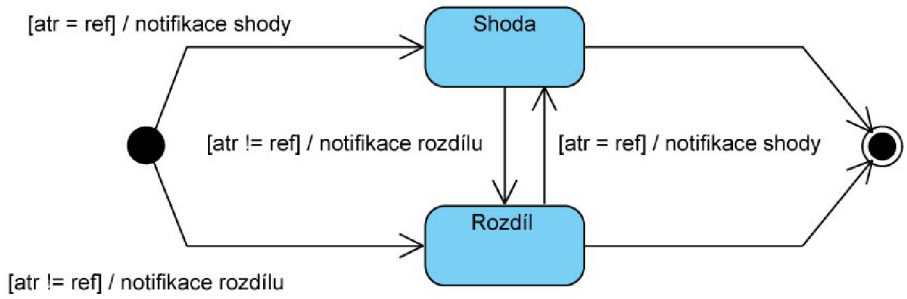
V rámci AMX jsou monitory definovány třídami `AMXStringMonitor`, `AMXCounterMonitor` a `AMXGaugeMonitor`. Následující tři sekce jsou věnovány popsání specifických vlastností jednotlivých typů.

7.3.2 Monitorování řetězců

Řetězcový typ monitoru reprezentuje AMX proxy `AMXStringMonitor`. Jeho princip je postaven na porovnávání hodnoty vybraného atributu s referenčním řetězcem. Notifikace, které generuje, upozorňují jak na shodu, tak na rozdíl obou hodnot. Stavový diagram na obrázku 7.1 ukazuje způsob monitorování řetězcového monitoru.

Navíc oproti společnému základu u řetězcového monitoru nastavujeme hodnotu řetězce pro porovnání metodou `setStringToCompare` a příznaky zasílání notifikací při shodě a při rozdílu hodnot metodami `setNotifyMatch` resp. `setNotifyDiffer`.

Tento druh monitoru můžeme využít všude v místech, kde chceme sledovat nějakou textovou hodnotu. Číselné ani jiné datové typy kontrolovat nelze. Porovnává se metodou `equals`. Musíme si vždy dávat pozor na skutečnost, že monitor při porovnávání bere v úvahu velikost písmen.



Obrázek 7.1: Stavový diagram řetězcového monitoru.

7.3.3 Monitorování čítačů

Pro monitorování čítačů slouží AMX mbeana `AMXCounterMonitor`. V tomto případě hraje hlavní roli hodnota atributu mbeany fungující jako čítač[2] a prahová hodnota pojmenovaná `Treshold`, nastavená v monitoru. Obě hodnoty jsou porovnávány. Jakmile čítač dosáhne nebo překročí úroveň prahu, monitor generuje notifikaci. Abychom mohli sledovat čítač v pravidelných intervalech, obsahuje monitor atribut nazývaný `Offset`. O tuto hodnotu se zvýší práh při každém jeho překročení hodnotou čítače.

Mezi další atributy, které můžeme u `AMXCounterMonitor` ovlivnit jsou:

- `InitTreshold` - počáteční hodnota prahu,
- `DifferenceMode` - diferenční mód. Způsobuje, že je práh porovnáván s rozdílem aktuální a poslední známe hodnoty atributu,
- `Modulus` - hodnota, kterou používáme k nastavení maximální prahové hodnoty. Při jejím překročení je použit právě modulus k operaci dělení modulo,
- `Notify` - příznak zasílání notifikací.

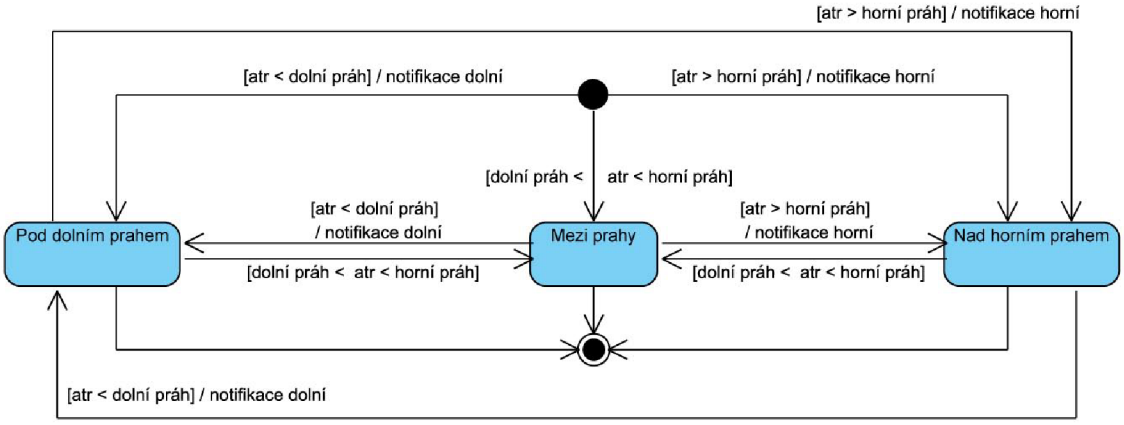
Při použití tohoto monitoru můžeme samozřejmě sledovat pouze číselné hodnoty a navíc jen ty celočíselné (`Byte`, `Short`, `Integer` a `Long`). Při nastavování sledovaného atributu monitor nekontroluje tuto podmínku. Problém nastává, až se monitor snaží hodnotu poprvé porovnat.

7.3.4 Monitorování rozsahů

Pro sledování rozsahů hodnot je připravena proxy `AMXGaugeMonitor`. Vše funguje na jednoduchém principu stanovení pevného rozmezí hodnot a následné kontroly obsahu atributu mbeany. Rozsah je definován dolním a horním prahem. `AMXGaugeMonitor` ukládá prahové hodnoty do atributů `LowTreshold` a `HighTreshold`. Stavový diagram na obrázku 7.2 ukazuje způsob monitorování rozsahového monitoru.

Stejně jako u monitorování čítačů se vždy kontroluje překročení prahů. U tohoto monitoru se však počítá se změnou hodnoty atributu oběma směry a u každého prahu se detekuje překračování pouze jedním směrem. To znamená, že jsou notifikace zasílány pouze pokud hodnota atributu sestoupí pod úroveň dolního prahu nebo naopak vystoupí nad úroveň prahu horního. V ostatních případech „křížení“ není vyvolána žádná reakce.

[2] Za čítač považujeme postupně inkrementovanou číselnou hodnotu pro počítání významných jevů.



Obrázek 7.2: Stavový diagram rozsahového monitoru.

Atributy pro konfiguraci `AMXGaugeMonitor` proxy jsou:

- `DifferenceMode` - diferenční mód označuje, že budeme kontrolovat rozdíl aktuální a poslední snímané hodnoty místo samotné hodnoty atributu,
- `Tresholds` - hodnoty dolního a horního prahu,
- `NotifyLow` - příznak zasílání notifikace pro dolní práh,
- `NotifyHigh` - příznak zasílání notifikace pro horní práh .

Na rozdíl od monitorování čítačů můžeme kontrolovat nejen celočíselné atributy, ale i atributy datových typů `Float` a `Double`. V oblasti webových služeb mají rozsahové monitory široké možnosti použití. Zvláště užitečné pak mohou být v případě kontrolování provozních statistik jednotlivých koncových bodů služeb (více o statistikách v sekci 7.2).

7.3.5 Využití monitorů

Použití monitorů v řídících aplikacích má své opodstatnění. Umožní nám sledovat hodnoty spravovaných prvků s určitou frekvencí a ihned reagovat na detekované změny stavů. Při bližším pohledu na princip fungování monitorů nelze nezjistit, že se nejedná o žádný převratný koncept. Monitory lze nahradit přípravou vlastních aktivních mbean pracujících na základě pravidelných podnětů od časovače. Tím spíše si uvědomujeme, jakou úsporu úsilí nám poskytují.

Při využívání AMX monitorů v řídících aplikacích jsme ale omezeni několika fakty. Hlavním požadavkem na bezchybné používání je správné nastavení monitorů. Důležité je nastavení intervalu kontroly dat. Musíme zvolit ideální hodnotu tak, aby se nám neztrácela informace mezi jednotlivými sběry vzorků dat. Na druhou stranu nesmíme zahltit server zbytečně vysokou frekvencí sledování. U monitorů rozsahu a čítačů hraje hlavní roli nastavení správných úrovní prahů. Dalším nedostatkem monitorů je možnost kontroly pouze jednoduchých datových typů. Nelze kontrolovat složitější datové struktury.

Kapitola 8

Aplikace pro řízení webových služeb

8.1 Úvod

Aplikace pro řízení webových služeb vznikla za účelem ověření postupů tvorby řídících systémů postavených na Java Management Extensions. Její implementací se snažíme otestovat, zda-li můžeme použít technologii JMX pro konkrétní oblast platformy Java EE, webové služby. Vzhledem k účelu, ke kterému byla připravována, ji však nelze považovat za komplexní řešení oblasti řízení a monitorování webových služeb. Informativní schéma aplikace je zobrazeno na obrázku 8.1.

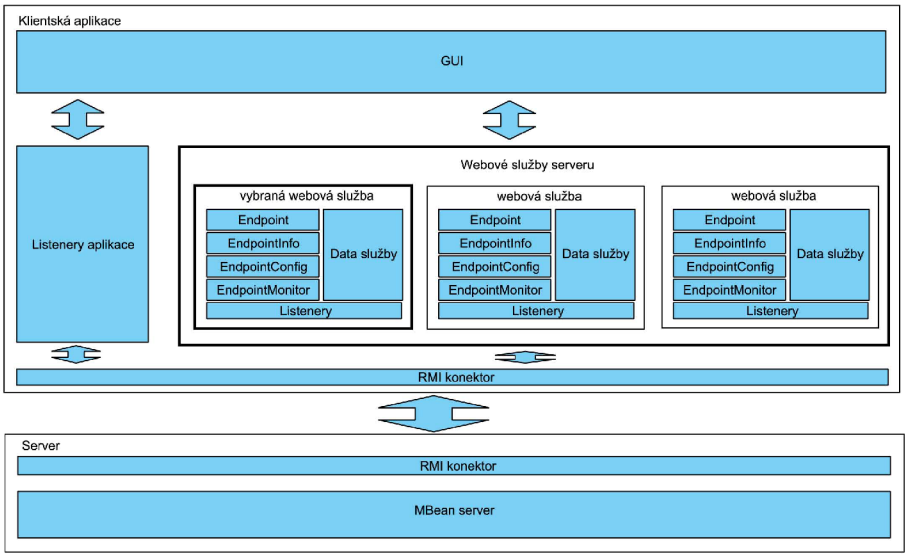
Prvotní myšlenkou bylo držet se pouze čistého JMX a nevyužívat nadstandardní prvky specifické vybranému serveru. Navzdory tomuto faktu jsme se později rozhodli zahrnout techniky dostupné jen na využívaném aplikačním serveru GlassFish[1]. Konkrétně šlo o aplikační rozhraní AMX (více v kapitole č.7). K takovému rozhodnutí nás motivovalo značné usnadnění práce kdy rozhraní AMX a jeho proxy objekty obstarávají mnoho elementárních a často opakujících se kroků nutných ke zprostředkování komunikace mezi serverem a naší aplikací. Druhou a neméně významnou motivací byla celková podpora JMX na serveru. Zde máme na mysli hlavně připravené generické mbeany webových služeb použitelné nezávisle na instrumentaci. Ve výsledném řešení jsme se však nevyhnuli kombinaci AMX rozhraní s nepřímým přístupem přes aplikační rozhraní JMX.

Tato kapitola slouží k provedení čtenáře postupy použitými při konstrukci aplikace. Snaží se zmínit to nejpodstatnější z implementace, vyzdvihnout základní části a principy, jaké byly aplikovány. Součástí je i popis nedostatků, na které jsme v průběhu implementace narazili. Kapitola si neklade za cíl vysvětlit do detailu vše z implementace.

8.2 Testovací služby

Abychom mohli ověřit správnost implementace naší aplikace, bylo potřeba připravit několik webových služeb. Přípravou rozumíme proces vytvoření klasické webové služby v Javě a její následnou instrumentaci řídícím rozhraním tak, abychom ji mohli aplikací sledovat a ovládat. Snažili jsme se webové služby připravit takovým způsobem, aby obsahovaly atributy, operace a vysílaly testovací události. Nakonec jsme tyto služby umístili na server.

[1] Používali jsme server GlassFish v2.1 vydaný v lednu 2009. Detaily této verze jsou popsány na URL http://wiki.glassfish.java.net/Wiki.jsp?page=PlanForGlassFishV2.1



Obrázek 8.1: Informativní schéma řídící aplikace

Standardní mbeana

Při konstrukci služby jsme umístili veškerou její logiku do samostatné třídy. Tato třída posloužila i pro implementaci řídícího rozhraní mbeany. Obsah rozhraní tak byl tvořen metodami pro službu i pro její řízení. Řídící rozhraní vzniklo podle názvu třídy a umístili jsme do něj hlavičky metod pro ovládání služby. Nakonec jsme také připojili logiku registrace a odregistrace formou anotovaných metod (anotace `@PostCreate` a `@Predestroy`) tak, aby registování i odstraňování mbeany reflektovalo vznik a zánik služby na serveru.

Dynamická mbeana

Při přípravě služby s dynamickou mbeanou jsme postupovali podobným způsobem. Rozdíl byl v tom, že namísto tvorby a implementace vlastního řídícího rozhraní jsme implementovali připravené rozhraní `DynamicMBean`. Metodami rozhraní jsme obstarali manipulaci s atributy a volání operací služby. Největší úsilí vyžadovala konstrukce metadat mbeany. Při jejich konstrukci jsme se řídili schématem z obrázku 6.1. Zajištění registrace a odregistrace bylo shodné se standardní mbeanou.

Notifikace

Zajištění vysílaní notifikací probíhá stejným způsobem u obou typů mbean. Za tímto účelem jsme implementovali rozhraní `NotificationBroadcaster` s využitím třídy `NotificationBroadcasterSupport`. Zaslání notifikace jsme iniciovali na základě změny hodnot jednoho z atributů. Pro objekt nesoucí detaily události jsme použili standardní třídu `AttributeChangeNotification`.

8.3 Základní funkce aplikace

Jak jsme již zmínili v úvodu, celá aplikace je postavená s využitím aplikačního rozhraní AMX serveru GlassFish. Kde to bylo možné, využívali jsme AMX proxy objektů. V ostatních případech jsme museli přistupovat nepřímým přístupem přes JMX (více v sekci 6.3.1). Základní princip fungování aplikace:

1. Aplikace se připojí k MBean serveru na základě uživatelem zadaných přístupových údajů. Uloží údaje o spojení potřebné ke komunikaci s mbeanami.
2. Vytvoří struktury reprezentující webové služby. Tyto struktury zahrnují generické proxy objekty služby, data o případné instrumentaci a ostatní informace spojené se službou.
3. Zaregistruje příjem událostí ze strany serveru, aby mohla aplikace reagovat na změny.
4. Inicializuje grafické rozhraní podle připravených webových služeb.
5. Aplikace reaguje na pokyny uživatele přijaté událostmi GUI a na příjem události ze serveru. Informace vypisuje ve formě textových zpráv do textového pole ve spodní části okna.

Pro zpřehlednění zde můžeme uvést, v jakých částech programu využíváme aplikačního rozhraní AMX a kde je naopak aplikovanou pouze JMX:

- *AMX* - aplikačního rozhraní GlassFish serveru využíváme k získání informací z generických mbean aplikačního serveru. AMX slouží v aplikaci také při instalaci nových webových služeb. Zde používáme AMX proxy `DeploymentMgr`.
- *JMX* - pro práci řídícím rozhraním námi instrumentovaných webových služeb jsme nuceni použít čisté JMX. Vzhledem k faktu, že dopředu neznáme obsah řídícího rozhraní, generujeme GUI dynamicky na základě metadat mbean služeb. Demonstraci fungování notifikací i monitorovacích služeb také provádíme v rámci JMX.

8.4 GUI a ovládání

Grafické uživatelské rozhraní rozhraní je rozděleno na základní dvě části. První z nich je okno webových služeb. Okno slouží pouze k vytvoření základního přehledu o instalovaných webových službách. Zobrazuje se po startu aplikace a slouží jako výchozí bod k výběru konkrétní služby. Spolu se seznamem služeb je nabízena možnost vyvolání dialogu instalace nové služby.

Druhým oknem se dostáváme k informacím o vybrané službě. Data strukturujeme podle zdrojů a kategorií do čtyř hlavních záložek:

- *Obecné vlastnosti* - Zde jsou zobrazeny informace získané z proxy `WebServiceEndpointInfo` a týkají se hlavně samotné implementace webové služby. Zobrazení obsahu konfiguračních souborů je provedeno formou dialogového okna spouštěného tlačítkem vybraného souboru. Spolu s těmito informacemi je zde připraveno nastavování monitorovacích parametrů z proxy `WebServiceEndpointConfig`, které ovlivňuje zobrazení následujících dvou záložek.

- *Statistiky* - Záložka statistik monitorovacích proxy `WebServiceEndpointMonitor`. Zobrazené statistiky pro jednotlivé instance virtuálních serverů lze resetovat připojeným tlačítkem.
- *Komunikace* - Tato záložka slouží k zobrazení komunikace webové služby s klienty. Využívá uložených dat z `WebServiceEndpoint`. Komunikace je dělena podle příslušnosti k virtuálnímu serveru. Obsahy zpráv jsou zpřístupněny přes dialogová okna stejně jako konfigurační soubory.
- *Bonus management* - Poslední záložka je spojena s informacemi získanými čistě z námi instrumentovaných služeb. Pokud služba nezpřístupňuje řídící rozhraní, záložka nemá s čím pracovat a je skryta. Opět informace z rozhraní strukturujeme do záložek, jejichž obsah modelujeme na základě metadat `MBeanInfo`:
  - *Atributy* - získávání a nastavování hodnot atributů,
  - *Operace* - volání metod s odpovídajícím počtem parametrů,
  - *Notifikace* - seznam notifikací přijatých od mbeany, možnost registrování i odhlášení jejich příjmu,
  - *Monitory* - prostor pro manipulaci s monitory. Zde můžeme vytvářet a nastavovat monitory v souvislosti s atributy mbeany služby.

Za účelem upozornění na zasílané události je ve spodní části aplikačního okna situováno textové pole. Sem se zobrazují informace o přijatých notifikacích v čitelné formě.

8.5 Objevené nedostatky

Instrumentace služeb

Při konstrukci webových služeb a jejich následné implementaci jsme nenarazili na vážnější problémy. Vše fungovalo tak, jak bylo popsáno ve specifikaci Java Management Extensions (přístupné v [8]). Jedinou otázkou, kterou jsme řešili byl životní cyklus mbean. Přesněji se jednalo o registraci a odregistraci mbean ze serveru spolu se službou. Nakonec jsme užili výše zmíněných anotací.

Generické mbeany

U generických mbean byl největší problém zjistit, jakým způsobem funguje jejich vytváření a rušení. V dokumentaci ([9]) není uvedena žádná zmínka o tomto chování, průvodce AMX ([11]) neuvádí také nic. Nakonec jsme tyto vlastnosti vysledovali až na základě přímého pozorování při jejich manipulaci.

Druhým problémem bylo jejich získání. Volání přístupových metod `getConfigPeer` a `getMonitoringPeer` mbeany koncového bodu nefungovalo vždy. Opět dokumentace ani průvodce neupřesňuje žádná omezení. Příčinu nefungujících metod se nepodařilo zjistit. Otázku získání mbean jsme vyřešili zvolením jiné přístupové metody (viz 7.1.2).

Obě záležitosti jsou detailněji probírány přímo u popisu generických mbean `WebServiceEndpointConfig` a `WebServiceEndpointMonitor` v sekci 7.2.

Monitorování

Při implementaci jednotlivých monitorů jsme postupovali cestou AMX proxy monitorů (popsáno v sekci 7.3). Pro manipulaci s monitory jsme používali mbeanu manažeru `JMXMonitorMgr`. U řetězcových monitorů fungovalo vše podle plánu. Manažer však odmítal odregistrovat mbeany ostatních dvou typů monitorů. Zůstávaly tak vždy „viset“ na serveru a znemožňovali opětovné vytvoření monitorů stejného jména. Po zdlouhavém hledání jsme našli zapomenuté dva zakomentované řádky ve zdrojovém kódu třídy manažeru. Nedostatek jsme nahradili vlastní odregistrací monitorů z MBean serveru.

Bohužel ani poté se nám nepodařilo zprovoznit systém monitorování přes AMX proxy. Při kontrole monitoru jsme neobjevili žádné chyby - monitor byl úspěšně registrován, spuštěn a pravidelně kontroloval nastavený atribut sledované mbeany. Nefungoval ovšem systém zasílání notifikací. Nakonec jsme u monitorů opustili rozhraní AMX a implementovali je složitější cestou nepřímého přístupu přes JMX.

Notifikace

U notifikací jsme se setkali s nedostatkem ze strany implementace mbean serveru GlassFish. Podle specifikace JMX musí každá mbeana vysílající události zveřejnit popis všech jejich typů metodou `getNotificationInfo` rozhraní `NotificationBroadcater`. Touto cestou se umožňuje potenciálním posluchačům mbeany identifikovat, co je vysíláno a na co lze reagovat. Bohužel jsme se však při práci s aplikačním rozhraním AMX téměř nesetkali s dodržováním této zásady. I přes jistotu zasílaných událostí tuto metodu většina AMX mbean nepřipravuje a nechává ji vracet prázdnou hodnotu.

Dokumentace

Jeden z hlavních nedostatků, se kterým jsme se setkávali po celou dobu práce s aplikačním rozhraním AMX, se týkal dokumentace (dostupné v[9]). Její obsah je v některých případech velmi strohý a nicneříkající. Většina popisů metod se omezuje na konstatování suchého faktu ve tvaru „vrací...“ nebo „vytvoří...“ bez dalšího vysvětlení. U některých metod i tento formát postačuje, u jiných ale potřebujeme znát detaily. Dokumentace má programátorovi pomáhat, ale u aplikačního rozhraní AMX tomu tak bohužel není.

Kapitola 9

Závěr

Cílem této diplomové práce bylo zkoumat možnosti webových služeb jako realizace architektury orientované na služby. Úkolem bylo zmapovat aktuální stav webových služeb a související rodiny standardů WS-* v rámci plnění hlavních myšlenek architektury. Pozornost byla kladena na otázky integrace, bezpečnosti, monitorování služeb a SOA governance. Bylo potřeba zkoumat jejich podporu webovými službami a definovat hlavní nedostatky jejich realizace. Dále byla požadována analýza prostředků platformy Java EE pro implementaci webových služeb včetně zmíněných součástí. Hlavním úkolem práce pak bylo najít nedostatky v některé ze zkoumaných oblastí a navrhnout možné řešení. Nakonec bylo potřeba navrhnuté řešení implementovat a ověřit možnosti jeho použití.

Architektura orientovaná na služby je logickým vyústěním procesu vývoje systémů v oblasti informačních technologií. Svými principy umožňuje vývoj rozsáhlých distribuovaných systémů efektivní cestou. Hlavními znaky těchto systémů jsou především pružnost reakcí na změny a schopnost integrace různorodých modulů.

Ačkoliv existuje více realizací SOA, webové služby se pro mnohé staly synonymem této architektury. Pokud uvažujeme realizaci hlavní komunikační infrastruktury formou jádra webových služeb, tvořeného standardy XML, SOAP a WSDL, nemůžeme ji téměř nic vytknout. Jedinou slabinou jádra webových služeb se tak stalo použití UDDI registrů. Samotná myšlenka registrů metadat služeb je správná. Chybnou se ukázala až koncepce přístupu k jejich obsahu. Standardy rodiny WS-*, které rozšiřují základní jádro webových služeb, poskytují mnoho prostředků pro zlepšení integrace, zvýšení bezpečnosti a realizaci SOA governance. Trpí bohužel nejednotností postupu ze stran producentů a celkovou absencí hlavní řídící autority.

Platforma Java EE prošla výrazným vývojem v oblasti webových služeb. Obsahuje mnoho nástrojů, umístěných do aplikačního rozhraní JAX-WS, pro komfortní realizaci služeb. Vše směřuje k čím dál vyšší abstrakci prostředků pro implementaci webových služeb. Programátor je tak odstíněn od zbytných detailů a soustředí se čistě na logiku aplikace.

Po analýze rozšíření webových služeb se dá prohlásit, že každé z nich je do jisté míry úspěšně realizováno. Za jednu z méně řešených otázek můžeme považovat oblast managementu a monitorování služeb náležící do SOA governance. Proto se tato práce soustředila na nalezení způsobu zajištění managementu a monitorování webových služeb.

Pro monitorování a management Java zdrojů slouží technologie Java Management Extensions. Způsob, jakým jsou v Javě implementovány webové služby se neliší od standardních vývojových postupů. Tento fakt se stává výhodou v souvislosti s možným použitím JMX pro management právě webových služeb. Po důkladném studiu základních principů JMX se ukázalo, že nic nebrání v řešení otázky managementu a monitorování služeb právě

touto technologií.

Java Management Extensions definují dva kroky pro správnou konstrukci a provoz řídícího systému. Prvním je zpřístupnění služby managementu formou instrumentace jejího kódu řídícím rozhraním. Při implementaci služeb jsme ověřili, že instrumentace řídícím rozhraním není problémem. Dají se aplikovat veškeré obecné postupy technologie i v případě webových služeb. Jediné problémy, spojené s registrací řídícího rozhraní, se nám podařilo úspěšně odstranit.

Druhým krokem je vytvoření klientské řídící aplikace. Pro ověření technik vývoje jsme implementovali elementární řídící aplikaci s využitím mnoha postupů definovaných standardem JMX. Při testování jsme využili námi upravených služeb spouštěných na aplikačním serveru GlassFish. Server poskytuje vlastní aplikační rozhraní AMX pro konstrukci řídících systémů. Snažili jsme se je uplatnit v maximální možné míře, abychom se vyhnuli implementaci náročnějších technik JMX. Implementace ukázala množství nedokonalostí AMX, kdy se i při dodržování elementárních pravidel uvedených v dokumentaci některé procedury chovaly zcela neočekávaně.

Zatímco prvky specifické serveru GlassFish vykazují množství nesrovnalostí a chyb, základní techniky konstrukce řídících systémů podle standardu JMX se v průběhu implementace demonstrační aplikace ukázaly jako bezproblémové. Hlavními nevýhodami tak zůstávají značný objem kódu nutný pro zabezpečení i těch nejjednodušších operací a omezené zdroje informací. Při studiu technologie JMX jsme se celou dobu potýkali s nedostatkem literatury. Oblasti obecného JMX se věnuje pouze několik anglicky psaných knih. Žádná z knih se nesoustředí na problematiku webových služeb.

Závěrem můžeme prohlásit, že s pomocí technologie Java Management Extensions jsme schopni konstruovat užitečné řídící aplikace určené pro správu v oblasti webových služeb. Potenciálním problémům můžeme zabránit, pokud se vyhneme specifikům aplikačních serverů a budeme striktně dodržovat standardy JMX.

Literatura

[1] Eric Newcomer: *Understanding Web Services*. Addison-Wesley, 2002, ISBN 0-201-75081-3.

[2] Eric Newcomer, Greg Lomow: *Understanding SOA with Web Services*. Addison-Wesley, 2004, ISBN 0-321-18086-0.

[3] J Steven Perry: *Java Management Extensions*. O'Reilly, 2002, ISBN 0-596-00245-9.

[4] Jindřich Štumpf: SOA - další fáze budování ICT? In *DATAKON 2006*, 2006, s. 77 – 92.
URL http://www.datakon.cz/datakon08/datakon2006_sbornik.pdf

[5] Martin Kuba: WEB Services. In *DATAKON 2006*, 2006, s. 93–112.
URL http://www.datakon.cz/datakon08/datakon2006_sbornik.pdf

[6] Petr Weiss, Marek Rychlý: Architektura orientovaná na služby, návrh orientovaný na služby, webové služby. [online], Naposledy navštíveno 11. 1. 2009.
URL http://www.fit.vutbr.cz/research/grants/index.php?file=%2Fproj%2F378%2FSOA_SOAD_WS.pdf&id=378

[7] Richard Monson-Haefel: *J2EE Web Services*. Addison-Wesley, 2003, ISBN 0-321-14618-2.

[8] Sun Microsystems Inc.: Java Management Extensions (JMX) Specification, version 1.4. [online], 2006.
URL http://java.sun.com/javase/6/docs/technotes/guides/jmx/JMX_1_4_specification.pdf

[9] WWW Stránky: Application Server Management Extensions Documentation. [online], Naposledy navštíveno 12. 5. 2009.
URL
https://glassfish.dev.java.net/nonav/javaee5/amx/javadoc/index.html

[10] WWW Stránky: Java Community Process. Naposledy navštíveno 12. 5. 2009.
URL http://www.jcp.org/en/home/index

[11] WWW Stránky: Sun Java System Application Server 9.1 Developer's Guide. [online], Naposledy navštíveno 12. 5. 2009.
URL http://docs.sun.com/app/docs/doc/819-3672?l=en

[12] WWW stránky: Managing and Monitoring Web Services in Project GlassFish. [online], Naposledy navštíveno 15. 1. 2009.
URL http://developers.sun.com/appserver/reference/techart/ws_mgmt.html

[13] WWW stránky: Service-oriented architecture. [online], Naposledy navštíveno 15. 1. 2009.
URL http://en.wikipedia.org/wiki/service-orientedarchitecture

[14] WWW stránky: Web services architecture. [online], Naposledy navštíveno 15. 1. 2009.
URL http://www.w3.org/TR/ws-arch/

[15] WWW stránky: XML. [online], Naposledy navštíveno 15. 1. 2009.
URL http://en.wikipedia.org/wiki/XML

[16] WWW stránky: The Java EE Tutorial. [online], Naposledy navštíveno 16. 5. 2009.
URL http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

Dodatky

Dodatek A. CD se zdrojovými soubory a textem práce

Dodatek B. Vzorový WSDL dokument

Dodatek C. Vzorová SOAP zpráva

Dodatek A

CD se zdrojovými soubory a textem práce

`aplikace` - archiv spustilené aplikace

`aplikace_zdroj` - zdrojové soubory aplikace

`dokumentace` - dokumentace aplikace ve formátu Javadoc

`služby` - archivy připravených webových služeb

`služby_zdroj` - zdrojové soubory archivů webových služeb

`text` - Textový dokument diplomové práce

`text_zdroj` - Zdrojové soubory k textovému dokumentu

`readme.txt` - Soubor s popisem obsahu CD

Dodatek B

Vzorový WSDL dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- Abstraktní část popisu služby -->
  <message name="GetBookPriceRequest">
    <part name="isbn" type="xsd:string" />
  </message>
  <message name="GetBookPriceResponse">
    <part name="price" type="xsd:float" />
  </message>

  <portType name="BookQuote">
    <operation name="getBookPrice">
      <input name="isbn" message="mh:GetBookPriceRequest"/>
      <output name="price" message="mh:GetBookPriceResponse"/>
    </operation>
  </portType>

  <!-- Konkrétní část popisu služby -->
  <binding name="BookPrice_Binding" type="mh:BookQuote">
    <soapbind:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getBookPrice">
      <soapbind:operation style="rpc"
       soapAction=
       "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
        <input>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </input>
```

```
        <output>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </output>
    </operation>
  </binding>

  <service name="BookPriceService">
    <port name="BookPrice_Port" binding="mh:BookPrice_Binding">
      <soapbind:address location=
       "http://www.Monson-Haefel.com/jwsbook/BookQuote" />
    </port>
  </service>

</definitions>
```

Dodatek C

Vzorová SOAP zpráva

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:proc="http://www.Monson-Haefel.com/jwsbook/processed-by">

  <!-- Hlavička obsahující metainformace -->
  <soap:Header>
    <proc:processed-by
     soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
     soap:mustUnderstand="1" >
      <node>
        <time-in-millis>1013694684723</time-in-millis>
        <identity>http://local/SOAPClient2</identity>
      </node>
      <node>
        <time-in-millis>1013694685023</time-in-millis>
        <identity>http://www.Monson-Haefel.com/logger</identity>
      </node>
    </proc:processed-by>
  </soap:Header>

  <!-- Tělo zprávy nesoucí informace ve formě XML dokumentu -->
  <soap:Body>
    <po:purchaseOrder orderDate="2003-09-22"
     xmlns:po="http://www.Monson-Haefel.com/jwsbook/PO">
      <po:accountName>Amazon.com</po:accountName>
      <po:accountNumber>923</po:accountNumber>
      <po:book>
        <po:title>J2EE Web Services</po:title>
        <po:quantity>300</po:quantity>
        <po:wholesale-price>24.99</po:wholesale-price>
      </po:book>
    </po:purchaseOrder>
  </soap:Body>
</soap:Envelope>
```