VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

TRANSAKČNÍ KOORDINACE SLUŽEB V JBOSS ESB

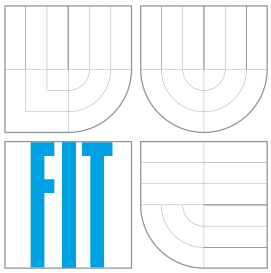
SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ ZELINKA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

TRANSAKČNÍ KOORDINACE SLUŽEB V JBOSS ESB

TRANSACTIONAL COORDINATION ACROSS SERVICES IN JBOSS ESB

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ ZELINKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK LETKO

BRNO 2012

Abstrakt

Tato práce se zabývá koordinací distribuovaných transakcí v prostředí jazyku Java. Cílem práce je navrhnout mechanizmus pro připojení transakčních zdrojů do aktivní distribuované transakce. V první části práce jsou vysvětleny základní principy transakčního zpracování a architektury orientované na služby. V druhé části se práce zaměřuje na popis konkrétních specifikací a použitelných nástrojů z projektu JBOSS od firmy RedHat. Po seznámení s principy a použitelnými prostředky je vytvořen návrh mechanizmu, který bude implementován a testován v diplomové práci.

Abstract

This project is focused on coordination of distributed transactions in Java language enviroment. The head point is proposal of mechanism for connecting transaction resources into active distributed transaction. The fundamentals of transaction processing and service oriented architecture are explained in the first part. The second part is focused on description of useful specifications and tools from project JBOSS, which is part of the RedHat company. The proposal of mechanism is described in the last part and it is going to be implemented and tested in master thesis.

Klíčová slova

Koordinace distribuovaných transakcí, Java Transakce, JBOSS ESB, JTA, WS-TX, WS-Coordination, SOA, SOAP, Java EE, webové služby

Keywords

Transactional coordination in distributed systems, Java Transactions, JBOSS ESB, JTA, WS-TX, WS-Coordination, SOA, SOAP, Java EE, web services

Citace

Tomáš Zelinka: Transakční koordinace služeb v JBoss ESB, semestrální projekt, Brno, FIT VUT v Brně, 2012

Transakční koordinace služeb v JBoss ESB

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pánů Ing. Zdeňka Letka a Jiřího Pechance. Uvedl jsem všechny literární prameny, ze kterých jsem čerpal.

.....
Tomáš Zelinka
10.ledna

Poděkování

Tímto bych velice rád poděkoval svým odborným vedoucím této práce Ing. Zdeňku Letkovi a Jiřímu Pechancovi za technickou podporu při vzniku celého projektu a technické zprávy.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá projektem JBoss ESB od firmy RedHat. JBoss ESB je projekt, který se používá při propojování více aplikací do jednoho fungujícího celku. Tomuto procesu slučování se říká systémová integrace. Při systémové integraci se slučují aplikace s různými komunikačními rozhraními a technologiemi, kdy tyto aplikace mohou být geograficky odděleny. Systémová integrace slouží k restrukturalizaci podnikových aplikací. Především slouží ke zvýšení efektivity jednotlivých aplikací, a tím zvýšením zisku celé společnosti. Při využití více aplikací je někdy potřeba zpracování více operaci z různých aplikací tak, aby při chybě nedošlo ke ztrátě dat nebo podobným bezpečnostním incidentům. K tomuto účelu je možno využít transakční zpracování. Transakce ze své podstaty nabízejí provedení jedné či více operací bezpečným způsobem. To znamená, že se transakce úspěšně provede nebo při výskytu chyby vrátí již provedené změny do původního stavu. Pokud jsou dílčí operace transakce prováděny na fyzicky oddělených systémech, jedná se o distribuovanou transakci, která vyžaduje koordinační mechanizmus. Koordinační mechanizmus distribuované transakce zabezpečuje dodržení správného postupu při provedení dílčích operací.

Hlavní cíl této práce je navržení a realizace koordinačního mechanizmu, který umožní vytvořit distribuovanou transakci pomocí JBoss ESB. Pro vytvoření mechanizmu budou použity některé specifikace, kterými se zabývá projekt JBoss Transactions. Dalšími cíli této práce jsou testování a demonstrace vytvořeného koordinačního mechanizmu na praktické aplikaci. Důležitou části této práce bude také objasnění pojmů souvisejících s vytvořením koordinačního mechanizmu.

Jako první jsou popsány základy architektury orientované na služby 2.1 a událostmi řízené architektury. Dále jsou objasněny obecné základy transakčního zpracování 2.3. Celá práce se zaměřuje na platformu programovacího jazyka Java a související přístupy k transakcím a podnikovému prostředí 2.5.

Další část práce se zaměří na JBoss produkty, které budou nebo mohou být využity v rámci práce. Bude popsána implementace podnikové sběrnice služeb od JBoos a specifikace JTA a WS-TX 3.2, které také jsou implementovány v JBoss a je možné je využít k vytvoření koordinačního mechanizmu. Tato část práce rovněž zahrnuje analýzu již implementovaných mechanizmů od jiných výrobců.

Třetí část obsahuje návrh implementace vlastního mechanizmu **??**. Tato část také obsahuje návrh testovací aplikace, která bude použita k testování a demonstraci vytvořeného mechanizmu koordinace **??**.

V další části **??** je popsána implementace testovací aplikace, která je založena na praktickém příkladu. V poslední části **??** je celá práce zhodnocena jsou diskutovány další možnosti distribuovaných transakcí, zvláště pak použití specifikace JTS.

Kapitola 2

Základní pojmy

V této kapitole je popsána většina důležitých pojmů a zkratek, které se vyskytují dále v práci nebo jsou důležité pro objasnění některého z použitých principů. Nejdříve budou popsány dvě základní architektury, kterých týká celá tato práce. Dále budou popsány principy transakcí a jejich rozšíření na distribuované systémy. V poslední části základních pojmů jsou popsány součásti platformy programovacího jazyka Java.

HTML

HTML je zkratka pro Hypertext Markup Language [17], což je značkovací jazyk pro vytváření strukturovaných dokumentů na webu. HTML umožňuje vkládat sémantické značky, které jsou zpravidla párové. Mezi páry značek tzv. tagů je umístěn text, který je na základě značek interpretován webovým prohlížečem

HTTP

HTTP je zkratka pro Hypertext Transfer Protocol [4]. Je to protokol pro výměnu hypertextových dokumentů ve formátu HTML. Výměna dokumentů probíhá formou dotaz-odpověď.

XML

XML je zkratka pro Extensible Markup Language [5]. Je to obecný značkovací jazyk, kterým je možno vytvářet konkrétní značkovací jazyky. Používá se pro přenášení dat mezi webovými aplikacemi a vytváření dokumentů. Při vytváření dokumentů se pomocí značek označuje význam textu, což hraje velkou roli např. při vyhledávání informací v dokumentu.

SOAP

SOAP je původně zkratka pro Simple Object Acsess Protocol, ale dnes se používá pouze SOAP jako samostatný pojem [16]. Specifikuje protokol pro výměnu strukturovaných a typovaných informací v decentralizovaných a distribuovaných sítích. Používá XML pro definování sady funkcí pro vytváření a výměnu zpráv nad různými podpůrnými protokoly. SOAP byl vyvinut tak, aby byl nezávislý na transportních protokolech.

Volně svázaný systém

Volně svázané (loosely coupled)[8] systémy obsahují na sobě nezávislé komponenty, které spolu komunikují a tak vytvářejí požadovaný systém. Nezávislost těchto komponent spočívá

v možnosti jejich obměny. Ve volně svázaném systému se mohou jednotlivé komponenty uvnitř měnit(verze, implementace, atd.) aniž by to negativně zasáhlo daný systém.

Těsně svázaný systém

Těsně svázané (tightly coupled)[8] systémy obsahují na sobě závislé komponenty, jejichž implementace je spojuje dohromady. V těsně vázaných systémech si nemůžeme dovolit měnit(implementace, verze, atd.) pouze jednu komponentu, aniž by to mělo negativní následky na chod ostatních komponent nebo systému jako celku.

Webové služby

Webová služba je softwarový systém vytvořen k podpoře interakce mezi uzly sítě[11]. Má rozhraní popsané strojově zpracovatelným formátem. Ostatní systémy komunikují s webovými službami způsobem předepsaným v popisu za použití SOAP zpráv založených na XML, které jsou přenášeny pomocí protokolu HTTP za použití serializace XML ve spojení s ostatními webovými standardy. Pro webové služby byla vytvořena řada specifikací, které mají ve svém názvu prefix WS-. Je možné se setkat s označením skupina specifikací WS-*.

Middleware

V distribuovaných systémech se jedná o vrstvu, která leží mezi operačním systémem a aplikací na každé straně systému. Hlavním úkolem middleware je skrytí distribuovanosti a různorodosti jednotlivých částí systému.[19]

Otevřený software

Otevřený software (open-source software)[3] je software s otevřeným zdrojovým kódem. Otevřenost znamená možnost zdrojový kód využívat při splnění určitých podmínek.

2.1 Architektura orientovaná na služby

Architektura orientovaná na služby(Services oriented architecture, SOA) [8] je sada principů a postupů pro návrh a vývoj softwaru. SOA se zaměřuje na přesně definované softwarové komponenty (části počítačových systému, datové struktury, apod.), které mohou být použity pro vícekrát, pro více účelů a v kontextu SOA sem jim říká služby. Nyní si si ukážeme hlavní principy a postupy, které se používají při vytváření služeb v rámci SOA.

Zapouzdření operace

Při vytváření systému obsahujících služby, rozsah zapouzdření může být menší i větší. Služba může zapouzdřovat jednu operaci v rámci systému nebo operaci, která reprezentuje celý systém.

Vztah mezi službami

Jednotlivé služby mohou být využívány jinými službami nebo programy. Aby mohly služby a programy mezi sebou komunikovat, musí o sobě vědět. Toto povědomí je realizováno pomocí popisu služby. V popisu služby je uveden název služby a druh dat, které služba očekává a vrací.

Komunikace mezi službami

Služby komunikují pomocí zpráv. Jakmile je zpráva odeslána, služba nad ní ztrácí kontrolu. Podle SOA by měla být zpráva nezávislou komunikační jednotkou, a tak zastávat stejnou pozici při komunikaci jako služba. Jinými slovy to znamená, že zpráva si sama určí službu, ke které bude doručena. Abychom toho dosáhli, je nutné vybavit zprávu patřičnou inteligencí. Jedním z možných přístupů SOA, kterým lze SOA implementovat, jsou webové služby. Nutno poznamenat, že ne všechny systémy implementované pomocí webových služeb splňují koncept SOA.

SOA pomocí webových služeb

Co jsou to webové služby jsme si popsali již dříve. Nyní se podíváme na jejich strukturu a jak pomocí nich lze implementovat principy SOA.

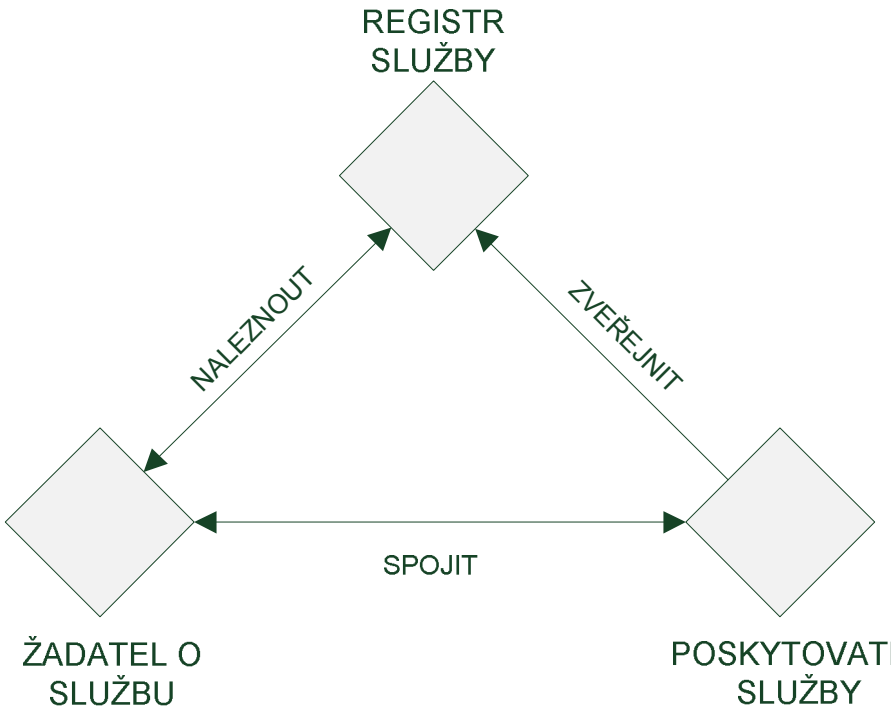
Zapouzdření operace do služby je zde realizováno právě pomocí webové služby. Popis služby lze realizovat pomocí jazyka pro popis webových služeb (WSDL, Web Service Definition Language). Komunikace je realizována pomocí SOAP protokolu. SOA pomocí webových služeb obsahuje 3 role :

- **Poskytovatel služby - Service provider**
  Je zodpovědný za popis služby a její spuštění na serveru. Pak je služba dostupná ze sítě a je možno zaregistrovat její popis v registrech, kde bude k nalezení pro žadatele o službu. Tuto roli lze připodobnit k serveru v architektuře klient-server.
- **Žadatel o službu - Service requester**
  Vyhledá požadovanou službu v jednom nebo více registrech. Po nalezení popisu dané služby se žadatel spojí se službou u poskytovatele a může ji používat. Tuto roli lze připodobnit ke klientovi v architektuře klient-server.
- **Registr služby - Service registry**
  Registr vystavuje popisy služeb dostupných od poskytovatelů pro žadatele, kteří v těchto registrech vyhledávají požadované služby. Jakmile je jednou služba v registru nalezena, není již tato role potřebná.

Tyto role mohou hrát jakékoliv programy nebo uzly v síti. Webové služby také obsahuje tři operace, které definují vztahy mezi jednotlivými rolemi:

- **Zveřejnění**
  Je vztah operace mezi registrem služeb a poskytovatelem, kdy poskytovatel zveřejní popis služby v registru, a tak ho zpřístupní žadatelům. Konkretní mechanizmus zveřejnění závisí na implementaci registru. V určitých případech roli registru přejímá sama síť, kdy je popis služby zveřejněn přímo v přístupné části souborového systému na serveru.
- **Nalezení**
  Žadatel definuje vlastnosti transakce a podle nich je vyhledána služba v registrech. Výsledkem hledání je seznam služeb, které odpovídají kriteriím. Nejjednodušší vyhledávací dotaz je HTTP GET[1] bez parametrů, který vždy vrátí všechny služby, které

[1] Hypertext transfer protokol - protokol pro přenášení hypertextových HTML dokumentů
GET - jedna z metod HTTP pro získání webové stránky klientem ze serveru



Obrázek 2.1: Model SOA pomocí webových služeb.

jsou k dispozici. Pak je na žadateli, aby vybral správnou službu. Pro komplexní registraci a vyhledání služeb se používá standardu pro univerzální popis, vyhledání a integraci(UDDI, Universal descrition, discovery and integration).

- **Spojení**
  Ustanovuje vztah klient-server mezi žadatelem o službu a poskytovatelem. Tento vztah může být dynamický, kdy mohou být za běhu vybírány jiné implementace dané služby daných kriterii. Také může být statický, kdy programátor přímo implementuje způsob volání dané služby.

SOAP zpráva

Jelikož služby mezi sebou komunikují pomocí zpráv, je nutné, aby zprávy měly jednotný formát a služby používaly stejný transportní protokol. Formát zprávy musí být flexibilní a rozšířitelný. Z obrázku 2.2 je patrné, že SOAP zpráva se skládá ze 4 částí:

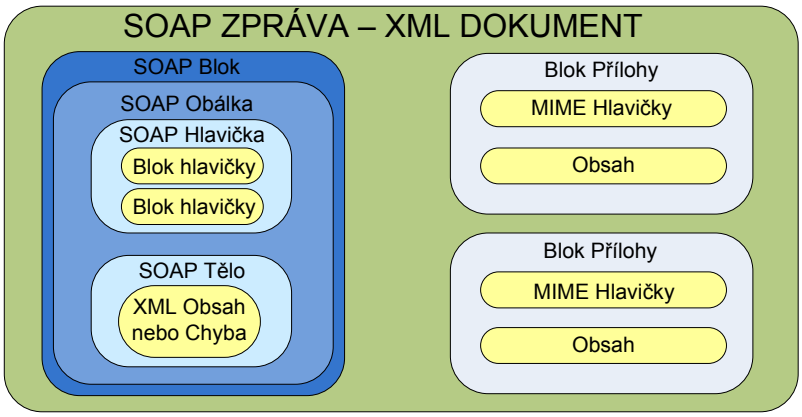
- **Obálka**
  Jak již název části napovídá, obálka se dá charakterizovat jako kontejner, který volitelně obsahuje hlavičku a povinně musí obsahovat tělo zprávy.

- **Hlavička**
  K dosažení zmíněné inteligence zprávy se využívá obsahu hlavičky SOAP zprávy. Obsah je složen z jednotlivých bloků, které mohou nést různé druhy informace související s doručováním zprávy, zpracování těla zpráv, bezpečností apod. Sémantika jednotli-

vých bloků není v SOAP specifikována. Každý blok může mít rozdílný význam, a tedy i specifikaci.

- **Tělo**
  Tělo zprávy obsahuje odesílaná data, které jsou ve formátu XML. V případě potřeby může zpráva obsahovat informaci o chybě na straně příjemce. Zpráva s informací o chybě je pak doručena zpět odesilateli.

- **Přílohy**
  SOAP příloha umožňuje posílat zejména data, která nelze popsat XML dokumentem. Obecně se jedná o binární data (multimédia, dokumenty, apod.).

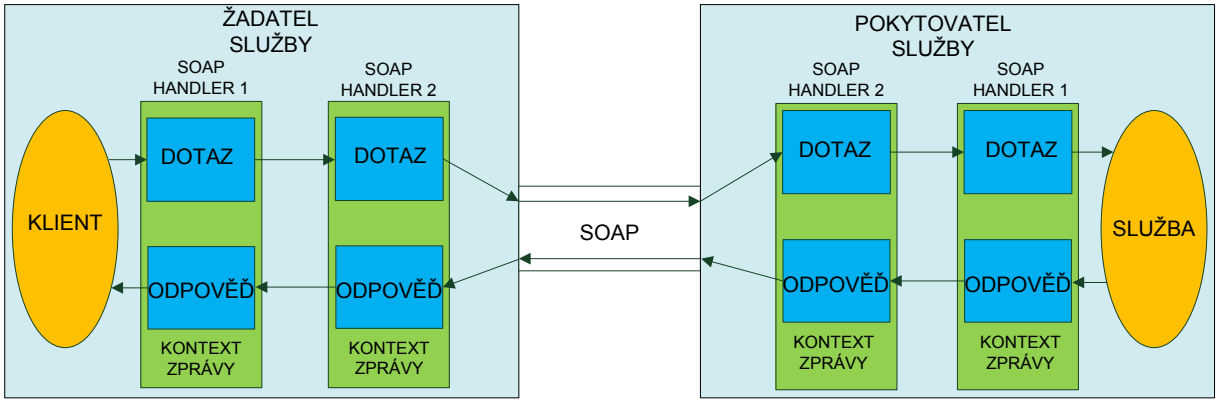


Obrázek 2.2: Obecný formát SOAP zprávy.

SOAP obslužné rutiny

SOAP obslužné rutiny (SOAP handlers)[11] mohou zpracovávat zprávu po příchodu a před odchodem zprávy. Jelikož se v hlavičce zprávy nacházejí různé informace(způsob zpracování těla zprávy, bezpečnostní kontext, transakční kontext, apod.), které je potřeba zpracovat dříve než je služba spuštěna, zpráva je nejdříve zachycena těmito rutinami a ty vykonají příslušné operace před spuštěním služby. SOAP obslužné rutiny se mohou nacházet na klientské straně i na koncové straně služeb.

Na klientské straně zpravidla přidávají informace (bloky hlavičky) do zprávy. Na koncové straně služeb čtou, případně odstraňují informace ze zprávy a vykonávají patřičné operace. Obslužné rutiny nemusejí být přítomny vůbec nebo jich muže být i více najednou.



Obrázek 2.3: Použití SOAP obslužné rutiny.

2.2 Událostmi řízená architektura

Událostmi řízená architektura (EDA, Event-Driven Architecture)[15] je další stupeň evoluce SOA[2]. Také je možné se setkat s pojmem událostmi řízená SOA (Event-driven SOA). EDA nabízí vyšší úroveň abstrakce oproti SOA. Stále se jedná o zprávami komunikující architekturu, jenž propojuje aplikace do volně vázaných systémů. Má však několik odlišností oproti SOA, které nabízejí další využití. Hlavní rozdíly oproti SOA:

Vztah mezi službami

SOA vyžaduje, aby jednotlivé služby měly o sobě povědomí pomocí zveřejnění jejich popisu. Toto v EDA již není třeba a služby mezi sebou mohou komunikovat aniž by znaly popis cílových služeb. Aby této vlastnosti bylo dosaženo je nutný přítomen mechanizmus pro manipulaci se zprávou, který ji transformuje během její cesty od odesilatele k příjemci.

Komunikace mezi službami

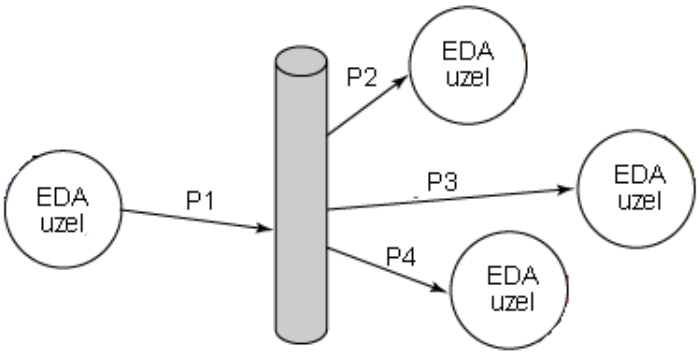
SOA je vhodná pro synchronní charakter aplikace typu dotaz/odpověď, kde je možno jednu stranu označit za klient a druhou za server. EDA je vhodná pro asynchronní dlouhotrvající procesy, kde může existovat více odesilatelů a více příjemců (komunikace many-to-many). Tato vlastnost je umožněna díky řízení pomocí událostí, kdy událost v kontextu EDA je možno charakterizovat jako významnou změnu stavu objektu (auto je na prodej → auto je prodáno).

Model podnikové sběrnice služeb

Podniková sběrnice služeb (ESB, Enterprise Service Bus) [6] je model, kterým je možno vytvořit systém reprezentující architekturu EDA. Základním úkolem ESB je propojit různé druhy aplikací (databáze, informační systémy, data, souborové systémy, webové služby) vytvořené různými technologiemi (Java, .NET, C/C++). Propojování aplikací, systémových komponent a různých subsystémů se v podnikové sféře nazývá *systémová integrace*. Jelikož

[2]EDA nenahrazuje koncept SOA, pouze ho rozšiřuje o další možnosti.

Obrázek 2.4: Architektura orientovaná na služby. [15]



Obrázek 2.5: Událostmi řízená architektura. P1-4 značí různé komunikační protokoly. [15]

je ESB pouze model, jeho implementaci mají v rukou vývojáři. V případě ESB se jedná o velice komplexní aplikaci, proto implementaci tohoto modelu nabízejí různí výrobci jako produkt (IBM, Oracle, RedHat aj.), kteří dosahují jednotlivých požadavků EDA různými technologiemi.

2.3 Transakce

Byly popsány základní architektury a modely, kterými se tato práce bude zabývat. Nyní budou popsány principy transakčního zpracování, které budou souviset s vytvořením mechanizmu pro koordinaci transakcí v ESB. Transakce je skupina operací, která se uživateli jeví jako jedna samostatná operace[13]. Operace v rámci transakce se musí provést všechny úspěšně nebo žádná. Síla transakce spočívá ve vykonání operací ve dvou fázích. V první fázi jsou výsledky jednotlivých operací transakce uloženy jako dočasný záznam mimo cíl a v druhé fázi jsou po výzvě transakce výsledky z toho záznamu přesunuty do cíle a dočasné záznamy jsou zrušeny. Pro zmíněnou výzvu transakce se používá výraz *commit*. Pokud dojde během první fáze transakce k chybě, jsou po výzvě transakce dočasné výsledky zrušeny. Při rušení výsledků se pro tuto výzvu používá výraz *rollback*. Po představení transakcí si řekneme proč transakce využít.

Transakce převádí systém z jednoho konzistentního stavu[3] do jiného konzistentního stavu. Jelikož konkretní implementace pojmů konzistentní stav a integritní omezení se napříč různými systémy (databázový, souborový, apod.) liší, ukážeme si na příkladu základní princip těchto pojmů a zároveň si představíme důležité vlastnosti transakcí.

[3]Konzistentní stav systému neporušuje žádné z jeho integritních pravidel

Mějme dva bankovní účty A a B. Na každém z nich je 50 korun, přičemž musí platit A + B = 100 korun.

Zde máme integritní omezení A + B = 100

Konzistence systému je porušena pokud součet obou účtů není roven 100 korunám a je kontrolována po každé manipulaci s účty. Zde máme některé ukázkové scénáře manipulací s účty:

- **Převod z jednoho účtu na druhý.**
  Převod je složen ze dvou operací, výběr z účtu A a vklad na účet B. Pokud by tyto dvě operace byly provedeny odděleně, hrozí nebezpečí ztráty peněz "někde"v systému např. díky výpadku proudu při jedné z operací. Transakce zaručuje provedení obou operací nebo vůbec žádné, proto nehrozí ztráta peněz během převodu. Tato vlastnost se nazývá atomicita.

- **Současně prováděné transakce.**
  Dva lidé současně přistupují k jednotlivým účtům a provedou současně převod z účtu A na účet B a obráceně. Nyní se dvě transakce pokusí manipulovat se stejnými daty. V tomto případě musí jedna transakce počkat na dokončení druhé. Pokud by ke stejným datům přistupovaly obě transakce současně a jedna by při první operaci selhala, hrozí porušení konzistence systému. Tomuto oddělení transakcí se říká izolace.

- **Výpadek systému.**
  Každé úspěšné dokončení transakce je trvale uloženo. Pokud nastane výpadek systému, tak po jeho obnovení je výsledek transakce opět dostupný. Tato vlastnost transakce se nazývá trvalost. Této vlastnosti nelze absolutně dosáhnout, protože existují i extrémní chyby, např. havárie pevného disku.

Transakce s výše uvedenými vlastnostmi atomicita, konzistence, izolace a trvalost (ACID, Atomicity, Consistency, Isolation, Durability) se nazývají atomické transakce nebo je možné se setkat s pojmem ACID transakce. Existují však transakce, které nemusejí všechny tyto vlastnosti zaručovat, protože pro daný charakter aplikace nejsou striktní ACID vlastnosti vhodné. Tento druh transakcí se využívá převážně v distribuovaných systémech a budou zmíněny později v textu.

2.4 Distribuované transakce

Do této doby jsme mluvili o lokálních transakcích v rámci jednoho systému. Jelikož se tato práce zabývá hlavně distribuovanými (globálními) transakcemi, vysvětlíme si v čem spočívají rozdíly mezi lokální a distribuovanou transakcí. Distribuovaná transakce je rozprostřena mezi několik systémů a ty mohou nebo nemusejí být přímo fyzicky odděleny. Aby byla transakce distribuovaná, postačuje rozložení jednotlivých operací v rámci transakce mezi různé systémy (databázový, souborový, logovací apod.) v rámci jednoho fyzického stroje. V kontextu distribuovaných transakcí se jednotlivé systémy nazývají *transakční zdroje*.

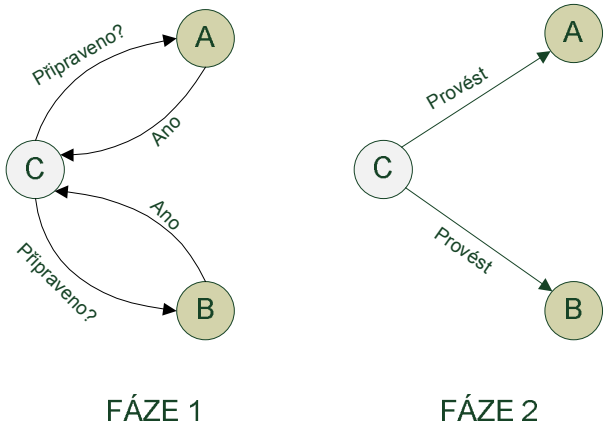
Pro zaručení ACID vlastností při distribuované transakci nepostačují pouze výzvy commit a rollback. Celá transakce se stává složitější, protože je nutné transakci synchronizovat a koordinovat. Koordinaci transakce má na starosti tzv. *transakční manažer*. Prozatím si

představme transakční manažer jako komponentu, která je zodpovědná za kontrolu transakce, a později v textu bude popsána jeho funkce detailněji. Transakční manažer potřebuje jistým způsobem komunikovat s různými systémy, k čemuž je potřeba jednotné rozhraní. Jednotné rozhraní pro kontrolu transakce poskytuje komponenta *Manažer transakčního zdroje*, která zapouzdřuje transakční zdroj[13].

Bylo zmíněno několik komponent, které jsou potřebné k realizaci distribuované transakce. Nyní se na jednotlivé komponenty podíváme detailněji a ukážeme si zjednodušený průběh distribuované transakce .

Manažer transakčního zdroje

Tato komponenta rozšiřuje daný transakční zdroj o funkce potřebné k transakčnímu zpracování. Je zodpovědná za komunikaci s transakčním manažerem i s transakčním zdrojem, který rozšiřuje. Manažer transakčního zdroje tedy vytváří vrstvu mezi transakčním manažerem a transakčním zdrojem. Zajišťuje ukládání dočasných výsledků transakce tak, aby bylo možné pracovat s transakčním zdrojem ve dvou fázích. Dále manažer transakčního zdroje rozšiřuje transakční zdroj o funkce obnovy po výpadku.



Obrázek 2.6: Obecně znázorněný Two-phase commit protokol. [13]

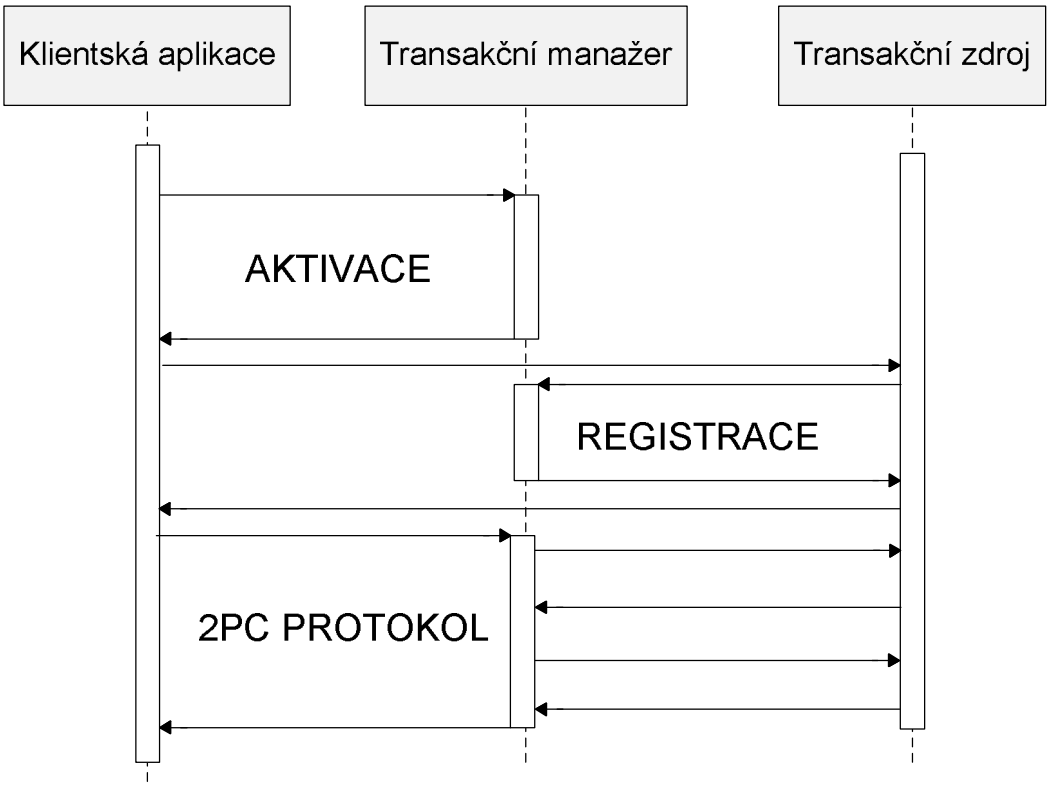
Transakční manažer

Transakční manažer je komponenta zodpovědná za koordinaci distribuované transakce. Často je označován také jako koordinátor. Jak bylo popsáno, lokální transakce jsou prováděny ve dvou fázích. U distribuovaných transakcí je situace velmi podobná. S tím rozdílem, že komunikace se může odehrávat na fyzicky oddělených strojích.

Zde již je třeba použít protokol, který zajistí správnou komunikaci s jednotlivými systémy a dvoufázové provedení transakce. Nejčastěji používaný protokol se nazývá dvoufázový commit protokol (2PC protokol, two-phase commit protocol).

Avšak provedení transakce není jediný úkol transakčního manažera. Jak je na obrázku 2.7 zjednodušeného průběhu distribuované transakce patrné, transakční manažer nejdříve

započne transakci (Aktivace), následně se u něj registrují jednotliví manažeři zdrojů (Registrace). Těsně po registraci dochází na straně transakčního zdroje k zablokování transakčního zdroje, provedení operace a uložení výsledků do dočasného úložiště. Transakční zdroj je nutné zablokovat, aby nedošlo k manipulaci jinou transakcí. Teprve potom dochází k dokončení transakce pomocí 2PC protokolu.



Obrázek 2.7: Zjednodušený průběh distribuované transakce.

2PC protokol

Je hlavním nástrojem pro zaručení atomického provedení distribuované transakce. Jeho úkolem je zajistit kooperaci všech zúčastněných systémů v transakci. Jak už z názvu vypovídá, protokol má dvě fáze. Existují různé modifikace tohoto protokolu s více fázemi, které slouží k vylepšení buď rychlosti zpracování nebo robustnosti transakce. Dvoufázový protokol je však nejpoužívanější.

- **Fáze 1**
  Během fáze 1 se koordinátor(C na obrázku 2.6) pokusí komunikovat se všemi transakčními manažery(A, B) transakce a zjistit, zda-li jednotlivé transakční zdroje jsou připraveny k dokončení transakce či nikoliv.

  Jakmile manažer transakčního zdroje dostane zprávu od koordinátora při fázi 1, vyhodnotí svou situaci a uloží si rozhodnutí o provedení nebo zrušení operace. Uložení

rozhodnutí se provádí pro případ neočekávaného výpadku transakčního zdroje a následné obnově konzistentního stavu. Každý manažer transakčního zdroje, který odešle kladnou odpověď, musí ponechat transakční zdroj zablokovaný, dokud neobdrží zprávu o fázi 2 od koordinátora.

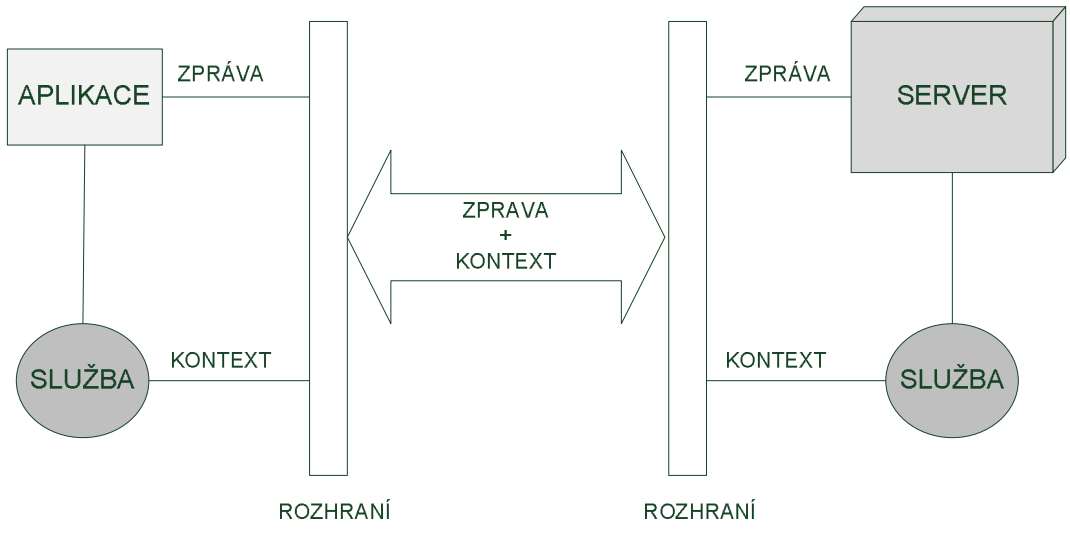
Pokud během kterékoliv operace došlo k chybě, je tato chyba oznámena koordinátorovi a celá transakce je zrušena (rollback). Při rušení transakce koordinátor uvědomí všechny manažery transakčních zdrojů, aby uvedli změny do původního stavu před započetím transakce.

- **Fáze 2**
  V této fázi koordinátor pouze vyzve všechny manažery transakčních zdrojů k dokončení transakce (commit). Na straně transakčního zdroje dochází k přesunutí výsledků operací do permanentního stavu a uvolnění zdroje.

Pro první fázi protokolu není nutné odesílat zprávy účastníkům sekvenčně. Je možné a kvůli efektivitě i žádoucí kontaktovat účastníky paralelně.

Transakční kontext

Jelikož jsou transakce distribuovány přes několik domén (transakční manažer, manažer transakčního zdroje, klientská aplikace), musí mezi nimi docházet k výměně určitých informací. Těmto informacím se obecně říká kontext. Kontext je propagován v distribuovaném prostředí, aby zajistil potřebný tok informací mezi danými doménami. Kontext vetšinou obsahuje identifikátor transakce, který je v dané transakci unikátní. Adresu nebo umístění koordinátora, aby se manažeři transakčních zdrojů mohli registrovat. Další specifické informace, například typ prováděcího protokolu (Two-phase, Three-phase apod.). Transakční systémy musejí obsahovat rozhraní (např. SOAP handler 2.3), které přidává kontext do odchozí zprávy a odebírá kontext z příchozí zprávy viz obrázek 2.8.



Obrázek 2.8: Přenášení transakčního kontextu. [13]

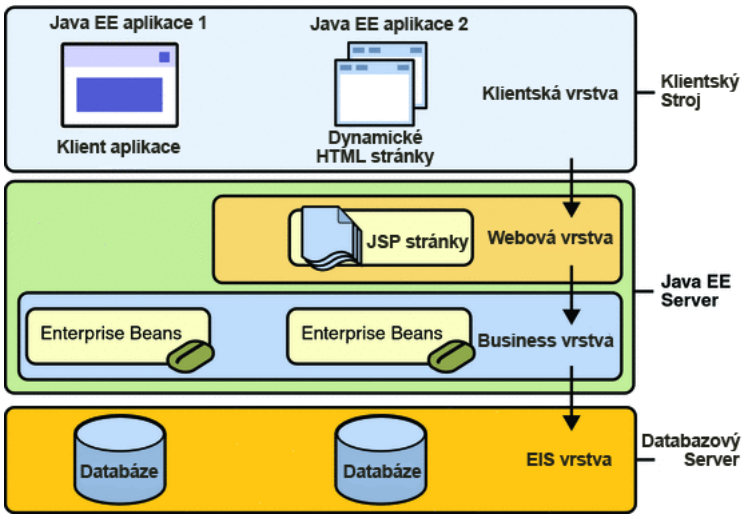
2.5 Platforma jazyka Java

Java Enterprise edition

Platforma Java poskytuje více druhů použití jazyka Java. Tato práce se zaměřuje na použití podniková edice Java platformy (Java EE, Java Enterprise edition)[12]. Existuje více částí platformy Java, které jsou určené pro různé použití. Java EE se používá pro vytváření rozsáhlých podnikových aplikací a informačních systémů. Základem Java EE je standardní edice jazyka Java (Java SE, Java Standard Edition), která se postupným vývojem rozšířila a musela být rozdělena na více edici včetně Java EE.

Aplikační logika Java EE je rozdělena do komponent podle funkce. Jednotlivé komponenty jsou umístěny v několika aplikačních vrstvách:

- **Klientská vrstva** - umístěna na straně klienta
  Tato vrstva typicky obsahuje uživatelské rozhraní pro ovládání cele aplikace.
- **Webová vrstva** - umístěna na straně Java EE serveru
  Na této vrstvě jsou umístěny komponenty vytvářející obsah, který je následně odeslán klientovi.
- **Business vrstva** - umístěna na straně Java EE serveru
  Zde je implementována aplikační logika, která přijímá vstupy od klienta, zpracovává je, případně je uloží do databáze.
- **Vrstva Podnikový informační systém(EIS)**
  Komponentami této vrstvy jsou databáze a podobné aplikace pro ukládání dat.



Obrázek 2.9: Jednotlivé vrstvy Java EE architektury. [12]

Takto navržené vícevrstvé aplikace je obtížné vytvářet, proto jsou komponenty vytvářeny tak, aby byly znovupoužitelné. Pak je mnohem snadnější vytvářet složité aplikace. Každá komponenta musí být umístěna do kontejneru. Kontejner je rozhraní mezi komponentou a funkcemi nižší úrovně, které jsou implementačně závislé. Obecným kontejnerem je aplikační server, který zprostředkovává služby ostatním kontejnerům a komponentám. Na aplikační

server je možné nasadit tzv. Enterprise JavaBeans (EJB)[4] a Web kontejner, do kterých lze umisťovat patřičné komponenty. Dále existují klientský a applet kontejner, které jsou vhodné pro klientskou část aplikace.

Pro vytváření vícevrstvých aplikací Java EE obsahuje velké množství technologii a jejich specifikací, které je možno rozdělit do několika kategorii:

- **Webové služby**
  Zahrnuje specifikace technologii pro implementaci SOA pomocí webových služeb. Například obsahuje specifikaci pro tvorbu webových služeb založených na XML (JAX-WS). JAX-WS je možné využít k tvorbě klientských aplikací a webových služeb založených na XML . Mezi klienty a službami lze vytvořit komunikaci založenou na zprávách nebo vzdálené volání procedur (RPC, Remote Procedure Call).

- **Webové aplikace**
  Zde jsou uvedeny technologie pro tvorbu webových aplikací a podporu pro implementaci modelu model-view-controller(MVC). Významnou specifikací z této kategorie je JavaServer Pages (JSP), která se používá k tvorbě dynamických webových stránek. Používá se společně se specifikací Java Servlet. Servlet je komponenta běžící v rámci webového serveru. Přijímá dotazy od klientů a generuje odpovědi ve formě HTML dokumentu. JSP slouží k oddělení uživatelského rozhraní od aplikační logiky. Pomocí JSP se vytváří HTML dokument s vnořenými úseky Java kódu.

- **Podnikové aplikace**
  V této kategorii jsou uvedeny technologie pro implementaci business logiky a různých podpůrných technologii. Pro vytváření podnikových aplikací je vhodná specifikace Enterprise JavaBeans (EJB). Pomocí EJB jsou vytvářeny komponenty, které oddělují aplikační logiku od prezentační vrstvy. Hlavní vlastnosti EJB komponent by měla být znovupoužitelnost. EJB komponenty je možno využít k zajištění bezpečnosti, transakčního zpracování nebo také k testování. Pro komunikaci mezi jednotlivými komponentami je vhodná specifikace Java Messaging Service (JMS).

- **Správa a zabezpečení**
  Obsahuje technologie pro monitorování a zabezpečení vytvořených aplikací. Pro tvorbu monitorovacích aplikací se používá specifikace Java Management Extensions (JMX). JMX umožňuje vytvářet aplikace s webovým rozhraním, které slouží k monitorování a správě různých zařízení, aplikací nebo sítí se službami. Aplikace v rámci JMX jsou vytvářeny pomocí Managed Bean (MBean) komponent, které zajišťují spojení se sledovaným objektem.

[4] EJB je kontejner, vhodný pro implementaci aplikační logiky

Kapitola 3

Projekty v rámci JBoss

JBoss je divize firmy Redhat, která se věnuje vývoji nových technologii v oblasti open-source middleware [7]. Tento projekt založil v roce 1999 Marc Fleury jako EJBoss (Enterprise Java Beans open-source software). V roce 2001 byl název zkrácen na nynější JBoss kvůli souvislostem s EJB od Sun Microsystems[1]. Od roku 2006 do současnosti je JBoss součástí firmy RedHat. JBoss je hlavně známý kvůli svému aplikačnímu serveru, který se stal velmi populárním. Jboss se věnuje i dalším technologiím, ke kterým patří i JBoss ESB a JBoss Transactions. V této části práce jsou JBoss ESB a JBoss Trsansactios popsány detailněji, protože součásti těchto projektů budou později použity při vytvoření koordinačního mechanizmu.

3.1 Jboss ESB

Projekt JBoss ESB se zabývá systémovou integrací pomocí modelu ESB. ESB je aplikační sběrnice, která má za úkol integrovat systémy založené na různých technologiích. Základem myšlenky ESB je oddělení aplikační logiky od transportních mechanizmů, transformace dat a dalších problémů, které nesouvisejí s aplikační logikou. Aplikační logika nebo celé systémy jsou zapouzdřeny do služeb a ty mezi sebou komunikují pomocí zpráv. Zprávy mezi službami jsou přenášeny po sběrnici, která je schopná je doručovat, směrovat, transformovat atd. Připojení jednotlivých technologii je možno obecně přes Java Connector Architekture (JCA). Dále je možno využít protokoly JMS, HTTP, SOAP a další, viz dokumentace [18].

Jboss ESB je vytvořen jako modul pro Jboss aplikační server. Abychom mohli vytvářet integrační aplikace,je potřeba tento modul nasadit[2] na aplikační server. Podobným způsobem se postupuje při tvorbě ESB aplikací. ESB aplikaci reprezentuje modul s příponou .esb, který se pak nasadí na aplikační server. V případě JBoss ESB modul představuje adresář se specifikovanou strukturou, jehož název musí obsahovat koncovku .esb. JBoss ESB aplikace mají dvě hlavní komponenty, zprostředkovatele (Providers) a služby (Services).

Zprostředkovatel

Zprostředkovatel (Provider) je komponenta zodpovědná za zprostředkování transportních služeb v rámci JBoss ESB. Zprostředkovatel umožňuje vytvořeným službám komunikovat mezi sebou nebo s externími službami. Právě pomocí zprostředkovatelů je možno připojit

[1] dnes již Oracle

[2] Nasazení není pouze nahrání modulu na aplikační server, ale zahrnuje i další úkony, které spojí modul s aplikačním serverem

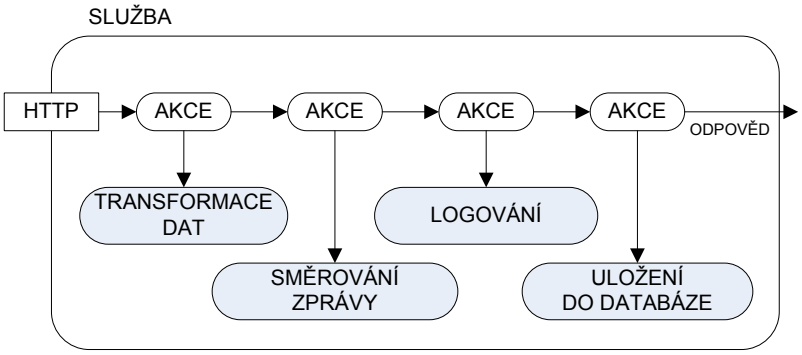
jednotlivé služby k transportním mechanizmům (např. fronta). Při konfiguraci je transportnímu mechanizmu přiřazen jednoznačný identifikátor, pomocí kterého jej bude moci některá ze služeb najít v registrech[3]. JBoss ESB podporuje komunikaci pomocí různých technologii (JMS, databáze, souborové systémy, apod.), aby bylo možné pokrýt co nejvíce možností integrace. V JBoss ESB existují dva základní typy zprostředkovatelů:

- **Událostmi řízený zprostředkovatel**
  První typ zprostředkovatele pasivně čeká na událost (např. příchod zprávy). Služba je spuštěna jako reakce na událost.
- **Zprostředkovatel řízený naplánováním**
  Druhý typ má zpracování události naplánován. Periodicky spouští služby, a tak zajistí zpracování zprávy. Tento druh je vhodný pro obsluhu systémů, které nejsou založené na událostech.

Služba

V Jboss ESB je služba tvořena tzv. *naslouchači* (listeners) a *akcemi* (actions). Při vyvolání služby jsou akce v rámci jedné služby postupně vykonány v pořadí, ve kterém byly nadefinovány. Toto uspořádání je pojmenováno zřetězení akcí *action pipeline*. K tomu, aby mohly být akce zřetězeny, musí být akce standardizovány a vykonat určité funkce. Hlavní funkcí každé akce je zpracování zprávy, která očekává zprávu a vrací zprávu zpět. Jinými slovy, každá akce převezme zprávu, zpracuje ji a předá ji další akci. Díky tomuto mechanizmu můžeme pomocí jednotlivých akcí transformovat data ve zprávách, zprávy můžeme směrovat do různých služeb, můžeme vytvářet záznamy o transportovaných datech atd.

Na obrázku 3.1 lze vidět příklad služby, která přijímá data pomocí HTTP. Po přijetí dat, je vytvořena vnitřní ESB zpráva, která zapouzdří data a zpráva putuje zřetězením akcí. Data jsou do zprávy ukládána asociativně, tedy každá položka dat má svůj klíč. Při vytváření jednotlivých akcí je nutno dát pozor, aby si při průchodu zřetězením jednotlivé akce neplánovitě nepřepisovaly data.



Obrázek 3.1: Zřetězení akcí - služba. [18]

Pro rychlejší vytváření služeb JBoss ESB disponuje připravenými akcemi (OOTB, Out Of The Box), které je stačí pouze nakonfigurovat. Také je možné vytvořit si úplně vlastní akci v jazyce Java. Abychom mohli využívat transportní mechanizmy nakonfigurované pomocí

[3] JBoss ESB používá UDDI registry k ukládání informací o službách, více v [18]

zprostředkovatelů, musíme službu propojit s připraveným transportním mechanizmem. To je možné díky naslouchačům. V JBoss existují dva typy naslouchače:

- **Zpracující externí požadavky** (Gateway listener)
  Přijímá data od externích služeb, které komunikují jiným způsobem než ESB zprávami. Po přijetí dat od externí služby zapouzdří data do ESB zprávy a přepošle jej naslouchači zpracující ESB zprávy.

- **Zpracující pouze ESB zprávy** (ESB-aware listner)
  Přijímá ESB zprávy odesílá jej do služeb. Zpráva může přijít od jiné služby v rámci ESB nebo může přijít od naslouchač externích požadavků, který transformoval data do ESB zprávy.

ESB zpráva

Protože zpráva je v JBoss ESB jediný komunikační prostředek, ukážeme si její strukturu a vlastnosti. JBoss ESB.

Služby mohou komunikovat způsobem dotaz/odpověď nebo jednosměrně. V prvním případě po odeslání zprávy čeká odesilatel předem určenou dobu na odpověď. Je doporučeno vytvářet komunikaci mezi službami jednosměrně, kdy dotaz a případná odpověď jsou na sobě nezávislé operace. Tento styl komunikace zvyšuje odolnost vůči chybám a naplňuje předpoklad volně svázaného systému.

Služby mohou komunikovat způsobem dotaz/odpověď nebo jednosměrně. V prvním případě po odeslání zprávy čeká odesilatel předem určenou dobu na odpověď. Je doporučeno vytvářet komunikaci mezi službami jednosměrně, kdy dotaz a případná odpověď jsou na sobě nezávislé operace. Tento styl komunikace zvyšuje odolnost vůči chybám a naplňuje předpoklad volně svázaného systému.

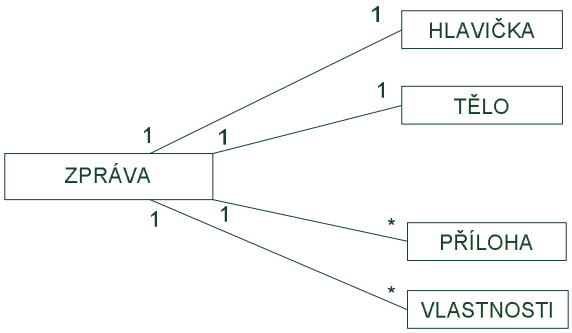
- **Hlavička** (Header)
  Hlavička obsahuje informace o směrování a adresaci . Zde je také umístěn jednoznačný identifikátor zprávy. Dále jsou zde umístěny informace, kam má zpráva putovat, pokud nastala chyba při zpracování.

- **Tělo** (Body)
  Data jsou do těla implicitně ukládány pod jednotným klíčem. Umístění dat, resp. název klíče, lze měnit uvnitř akce. Pro tělo zprávy jsou vytvořeny rozhraní pro manipulaci s textem, serializovatelným objektem, mapovaným objekt (klíč - serializovatelný objekt), bytovou sekvencí.

- **Příloha** (Attachment)
  Příloha se používá pro připojení dalších typů dat. Důvodem proč použít přílohy může být přehlednější struktura zprávy.

- **Vlastnosti** (Properties)
  Ve vlastnostech lze specifikovat ostatní informace ohledně zprávy, které není vhodné umístit do ostatních položek.

Není doporučeno používat Přílohy a Vlastnosti při implementaci, protože v budoucnu se bude jejich implementace měnit.



Obrázek 3.2: Struktura vnitřní ESB zprávy. [18]

3.2 JBoss Transactions I.

Jboss Transactions je projekt zabývající se implementací specifikací pro transakční zpracování. V První části je popsána specifikace Java Transaction API (JTA) a její implementace v rámci projektu JBoss Transactions. JTA je objektově orientované mapování na XA specifikaci. XA specifikace je technický standard pro distribuované transakční zpracování.

JTA je rozhraní k použití transakčního zpracování. Pro použití JTA je třeba mít spuštěný modul transakčního manažera na aplikačním serveru nebo je možné instanci spustit samostatně bez použití aplikačního serveru. Dále v textu je předpokládáno, že transakční manažer je spuštěn v rámci aplikačního serveru. Pro použití je třeba lokalizovat instanci transakčního manažera v registrech pomocí rozhraní pro pojmenování a adresáře (JNDI, Java Naming and Directory Interface). V rámci modulu transakčního manažera je spuštěno více služeb (pro obnovu po pádu transakčního zdroje - Recovery Manager, pro ovládání transakce na klientské straně - User Transaction, další viz dokumentace). Důležitou službou pro tvorbu aplikací s transakčním zpracování je služba User Transaction.

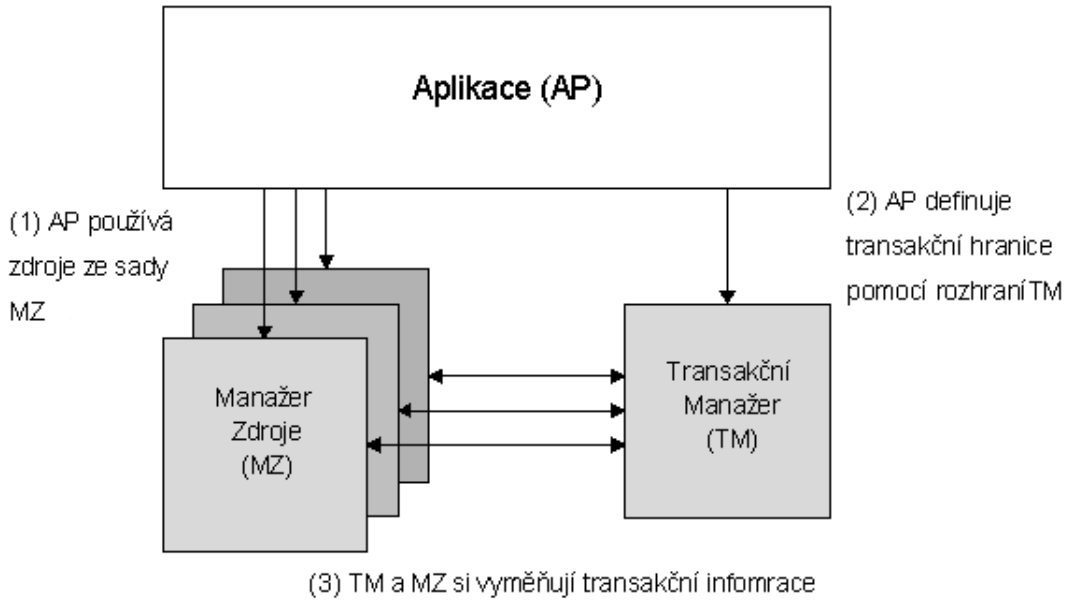
Aby bylo možné k transakci připojit různé druhy transakčních zdrojů, JTA dále poskytuje rozhraní pro komunikaci mezi transakčním manažerem a transakčním zdrojem (XAResource). Toto rozhraní musí daný transakční zdroj implementovat, aby mohl komunikovat s transakčním manažerem. Nyní budou popsány hlavní rozhraní, které jsou nezbytné pro vytvoření aplikace používající JTA.

Rozhraní UserTransaction

Toto rozhraní slouží k ovládání lokální transakce. Obsahuje metody pro zahájení (begin()), provedení(commit()) a zrušení transakcí(rollback()). Pro použití transakčního zpracování je nejdříve nutné zahájit transakci. Po zahájení transakce je třeba navázat spojení s transakčním zdrojem. Po započetí transakce se při navázání spojení automaticky transakční zdroj registruje u transakčního manažera.

Rozhraní XAResource

Toto rozhraní definuje vztah mezi transakčním manažerem a účastníky podle XA specifikace. Rozhraní je implementováno na straně transakčního zdroje. Transakční manažer vytváří instanci třídy XAResource pro každého účastníka, kterého chce přiřadit do trans-



Obrázek 3.3: Model distribuovaného transakčního zpracování podle XA specifikace.[1]

akce. Funkce *start()* a *end()* slouží k připojení a odpojení transakčního zdroje k vytvořené transakci . Transakční manažer na konci transakce pomocí metod *prepare()*, *commit()* nebo *rollback()* vykoná 2PC protokol, aby dokončil, případně zrušil transakci.

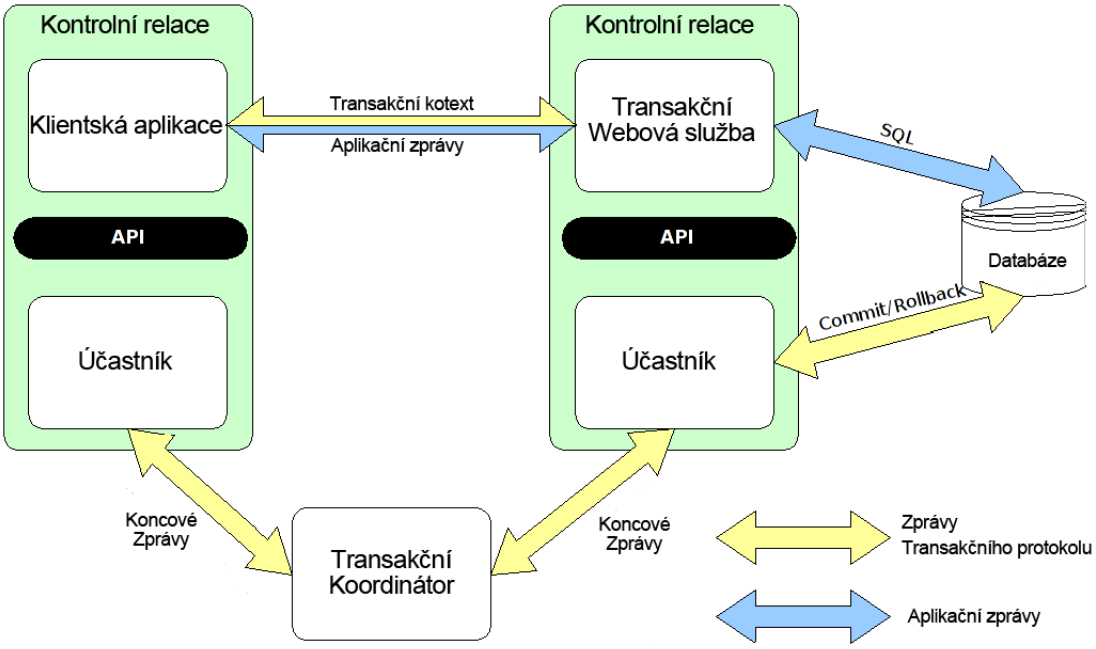
Distribuovanou transakci je také možné vytvořit bez účasti transakčního manažera právě díky XAResource rozhraní. Pokud získáme instanci XAResource objektu, můžeme využít funkce tohoto rozhraní a řídit transakci přímo v aplikaci.

Rozhraní TransactionManager

Rozhraní transakčního manažera je velice podobné rozhraní UserTransaction. Navíc pouze obsahuje funkce pro pozastavení (suspend()) a obnovu (resume()) transakce. JTA podporuje dočasné pozastavení a obnovení transakce, aby řídící vlákno mohlo vykonat nějakou jinou netransakční činnost. Pro pozastavení transakce se používá metoda *suspend()*, která vrátí ukazatel na objekt transakce a pozastaví transakci. Operace pozastavení prakticky transakci nepřeruší, pouze přeruší spojení transakce a vlákna. Takto je možné, aby jiné vlákno mohlo obnovit transakci, kterou nepozastavilo. Toto je vhodné pokud je třeba, aby více vláken pracovalo s jednou transakcí.

3.3 JBoss Transactions II.

V druhé části jsou popsány specifikace pro transakční zpracování pomocí webových služeb. Webové služby nabízejí řešení jedné z nevýhod JTA a to je spolupráce mezi různými technologiemi. Další rozdíl oproti JTA je fakt, že díky charakteru webových služeb můžeme rozdělit jednotlivé operace v transakcích na více virtuálních strojů. Tuto funkci JTA specifikace neumožňuje, pouze její rozšíření pro distribuované transakce s podporou pro komunikaci po síti (JTS, Java Transactions Service), která využívá standard pro komunikaci objektů běžících na různých strojích (CORBA, Common Object Requester Broker Architecture). Webové služby musí splňovat základní vlastnosti, které byly popsány v první části práce. Pro splnění těchto vlastností bylo vytvořeno několik specifikací. Pro transakční zpracování pomocí webových služeb byla vytvořena specifikace WS-Transactions (WS-Tx). Tato specifikace je tvořena specifikacemi WS-Coordination (WS-C) [9], WS-AtomicTransactions (WS-AT) [14] a WS- BusinessActivity (WS-BA) [10] [2]. Implementace WS-Tx v JBoss Transactions má označení XTS.



Obrázek 3.4: Model distribuovaného transakčního zpracování podle XTS.[18]

WS-Coordination

Specifikace pro vytvoření koordinátora jako služby, která umožňuje vytvoření koordinačního kontextu, registrace koncových účastníků a podporu více koordinačních protokolů. WS-C je možno použít k vytvoření koordinace jiných distribuovaných systémů než jsou transakční. Pro vytvoření koordinace transakčních systémů musí být WS-C doplněna o specifikace WS-AT a WS-BA.

Koordinátor má na starosti vytvoření koordinačního kontextu, registraci účastníků a provedení 2PC protokolu. Koordinátor specifikace WS-TX se skládá z několika základních komponent, pomocí kterých řídí transakci.

- **Aktivační služba**
  Je zodpovědná za vytvoření vytvoření koordinačního kontextu. Koordinační kontext obsahuje typ koordinace, jedinečný identifikátor transakce, dobu čekání na odpověď, typ prováděcího protokolu a další potřebné informace pro řízení transakce. Aktivační služba převezme od aplikace, která ji vyvolala, potřebné informace a vytvoří koordinační kontext, který je vložen do hlavičky SOAP zprávy.

- **Registrační služba**
  Jakmile přijme aplikace transakční kontext, začne registrovat účastníky do transakce. Registrace probíhá odesíláním zpráv registrační službě. Zpráva musí obsahovat informace o účastníkovi, který má být registrovaný, nejčastěji adresu účastníka a roli jakou má hrát v transakci.

- **Koordinátor**
  Koordinátor je zodpovědný za řízení komunikace mezi jednotlivými účastníky transakce. Dále pak řídí protokoly dokončující transakci. V JBoss transactions je implementováno více druhů koordinátorů. Každá specifikace (WS-AT, WS-BA) má svou implementaci koordinátora. Koordinátor je reprezentován webovou službou a komunikace je realizována SOAP protokolem.

- **Koordinační Kontext**
  Kontext obsahuje informace o transakci a jejich účastnících a je propagován mezi všechny zúčastněné služby.

WS-AtomicTransactions

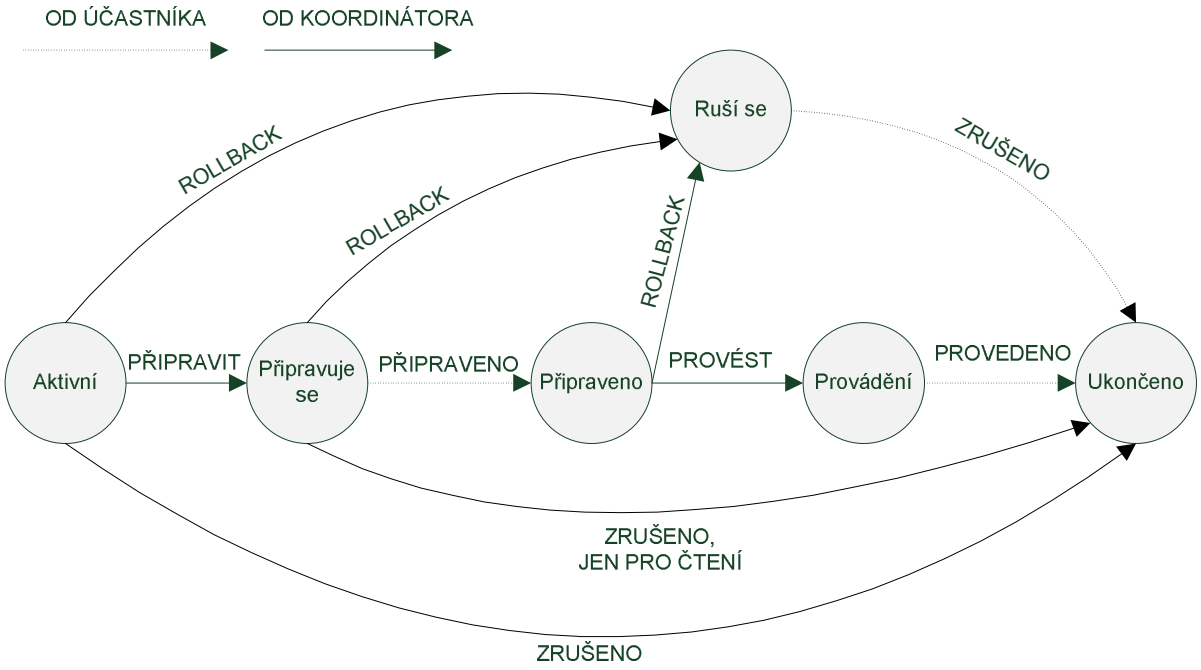
WS-AT doplňuje WS-C o koordinační protokol atomických transakcí. Atomické transakce pomocí webových služeb musí splňovat ACID vlastnosti 2.3. Pro zabezpečení ACID vlastností jsou ve specifikaci uvedeny 2 typy 2PC protokolu:

- **2PC nestálých zdrojů** (Volatile 2PC)
  Tento protokol není povinné používat. Protokol se používá při použití vyrovnávací paměť a je třeba ji vymazat před dokončením transakce. Vyrovnávací paměť se používá v transakcích pro vylepšení výkonu transakcí. Funkce protokolu je stejná jako při první fázi 2PC protokolu, kdy se zjišťuje, zda jsou účastníci připraveni. V tomto případě jsou možné 3 odpovědi: kladná, záporná a pouze pro čtení. Odpověď *Pouze pro čtení* se používá, pokud účastník neposkytuje transakční data, a tedy se nezúčastní druhé fáze protokolu. Při záporné odpovědi se transakce zruší a při kladné se pokračuje 2PC protokolem pro trvalé zdroje.

- **2PC trvalých zdrojů** (Durable 2PC)
  Dvoufázový prováděcí protokol, jehož obecný princip byl popsán v první části textu 2.6. Na obrázku níže vidíme jednotlivé stavy protokolu a možnosti, které mohou nastat v jednotlivých stavech. První fáze se dá charakterizovat jak přípravná a je zde největší pravděpodobnost zrušení transakce. Během příprav může koordinátor obdržet zprávu *pouze pro čtení*, což znamená, že daný účastník neposkytuje transakční data a bude tak vyřazen z druhé fáze protokolu aniž by došlo ke zrušení. Na první pohled není patrné jaký rozdíl je mezi přechody *rollback* a *zrušeno*. Pokud dojde k jakékoliv chybě během transakce, musí koordinátor zrušit transakci. Tuto situaci znázorňuje přechod *zrušeno*. Následně však koordinátor musí všem ostatním účastníkům rozeslat

zprávy, aby uvedli veškeré provedené změny do původního stavu, protože transakce byla ukončena.

Během druhé fáze není patrné, že by mohlo dojít k nějaké chybě. V této fázi se již předpokládá, že účastníci jsou připraveni provést dané změny a potvrdit je. Během tohoto procesu však může dojít k nečekaným komplikacím, například výpadek jednoho z účastníků. Takovéto chyby vedou k heuristickým[4] výsledkům transakce. Řešení takovýchto situací však nejsou součástí tohoto protokolu a jsou pro ně definovány speciální postupy [13].



Obrázek 3.5: 2PC protokol podle specifikace WS-Tx. [14]

WS-BussinesActivity

Ne všechny transakce musí dodržovat ACID vlastnosti, protože pro některé druhy aplikací nejsou vhodné. V některých případech není vhodné nebo možné blokovat transakční zdroj po celou dobu transakce, jak tomu je při atomických transakcí. Například při rezervaci zboží, které ještě není na skladě, není možné blokovat část databáze, dokud nepřijde zboží na sklad. Pro tento druh dlouho trvajících transakcí je vhodná tato specifikace. Oproti atomickým transakcím dlouho trvající transakce je možné dokončit a až po dokončení reagovat na chyby. To je vhodné např. při odesílání elektronické pošty, kdy chyba při doručení se projeví až s určitým zpožděním. Takové chování transakce musí mít podporu na straně účastníka. Při atomických transakcích účastník sdělil svůj stav pouze na dotaz koordinátora. Při Dlouho trvajících transakcích účastník informuje o vzniklé chybě samostatně bez dotázání. Při

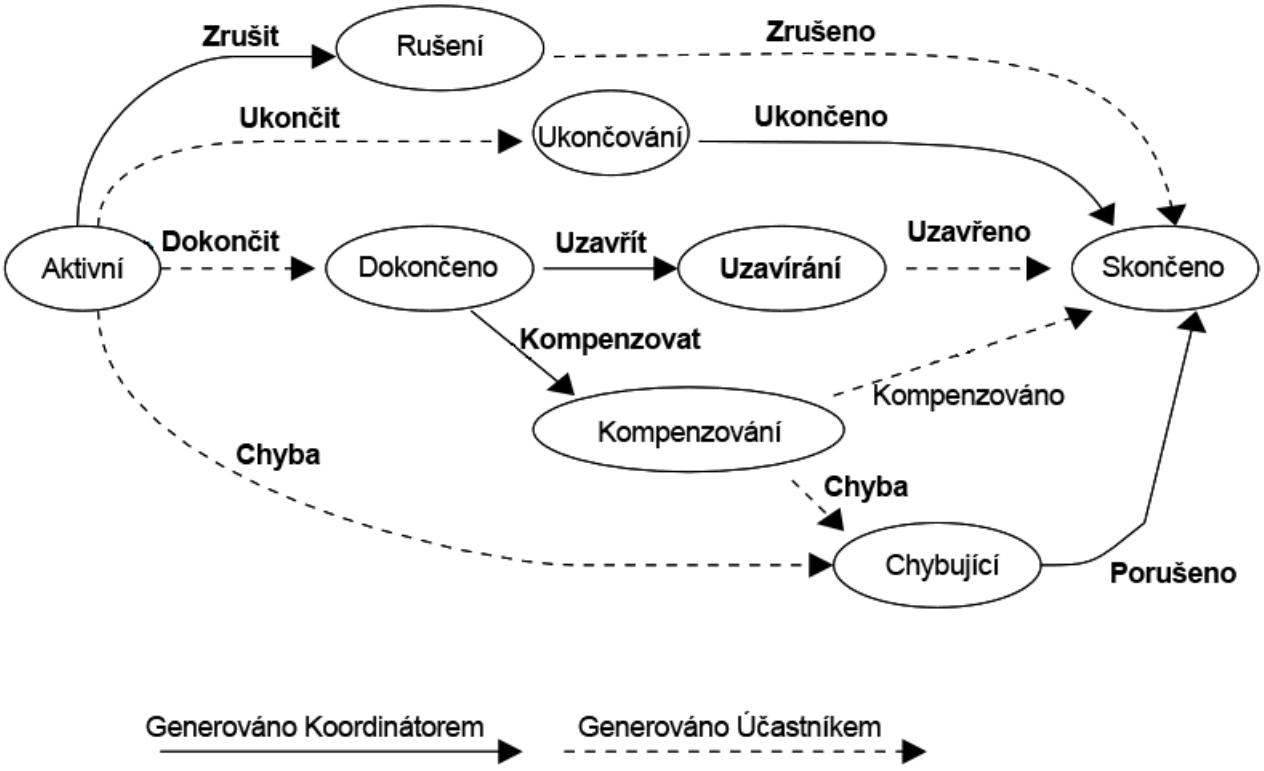
[4]Heuristický výstup transakce označuje pojem, kdy se během 2PC protokolu stane neočekávána chyba. Například koordinátor odeslal příkaz rollback, ale účastník z neznámého důvodu provedl příkaz commit.

vzniklé chybě se celá transakce vrátí do původního stavu, tak jako při atomických transakcí. Pro navrácení provedených změn se v kontextu dlouho trvajících transakcí používá pojem *kompenzace*.

Pro dlouho trvající transakce lze použít dva druhy protokolů provedení :

- **Provedení s dokončením účastníkem**
  (Business agreement with participant completion)
  Nejprve je účastník přidán do transakce a jeho počáteční stav je nastaven na *aktivní* . Po vykonání úkolu, pro který byl účastník přidaný do transakce, může vyslat koordinátoru zprávu, která signalizuje *ukončení* činnosti účastníka. Pokud již není třeba účastníka při transakcí může se ukončit a tak se odhlásit z transakce. Účastník také může vyslat zprávu signalizující ukončení činnosti, avšak čekající na příkaz ukončení od koordinátora. Koordinátor následně musí poslat příkaz úspěšnému dokončení transakce nebo ke kompenzaci provedených změn.

- **Provedení s dokončením koordinátorem**
  (Business agreement with coordinator completion)
  Tento protokol je velmi podobný předcházejícímu, pouze s tím rozdílem, že účastník nemůže bez příkazu koordinátora ukončit svou činnost a odhlásit se z transakce.



Obrázek 3.6: Business Activity protokol podle specifikace WS-Tx. [10]

Literatura

[1] *Distributed Transaction Processing: The XA Specification* [online]. The Open Group. 1991 [cit. 2012-05-23]. Dostupné z WWW: <http://pubs.opengroup.org/onlinepubs/009680699/toc.pdf>.

[2] *Web Service Transactions Programmers Guide* [online]. JBoss. 2012 [cit. 2012-05-23]. Dostupné z WWW: <http://docs.jboss.org/jbosstm/docs/4.2.3/manuals/html/xts/ProgrammersGuide.html>.

[3] *The Open Source Definition* [online]. Open Source Initiative. [cit. 2012-05-23]. Dostupné z WWW: <http://www.opensource.org/docs/osd>.

[4] BERNERS-LEE, T. - LEACH, P. - MASINTER, L. - aj. *Hypertext Transfer Protocol – HTTP/1.1* [online]. RFC 2616. 1999 [cit. 2012-05-23]. Dostupné z WWW: <http://www.ietf.org/rfc/rfc2616.txt>.

[5] BRAY, T. - PAOLI, J. - MALER, E. - aj. *Extensible Markup Language (XML) 1.1 (Second Edition)* [online]. 2006 [cit. 2012-05-23]. Dostupné z WWW: <http://www.w3.org/TR/2006/REC-xml11-20060816/>.

[6] CHAPPELL, D. *Enterprise Service Bus: Theory in Practice*. O'Reilly, 2004. 274 s. ISBN 0596006756.

[7] CONNER, K. - CUNNINGHAM, T. - DIMAGGIO, L. - aj. *Jboss Esb Beginner's Guide*. Packt Publishing, 2012. 323 s. ISBN 1849516588.

[8] ERL, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005. 792 s. ISBN 0131858580.

[9] FEINGOLD, M. - JEYARAMAN, R. *OASIS Web Services Coordination Version 1.2* [online]. 2. února, 2009 [cit. 2012-05-23]. Dostupné z WWW: <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html>.

[10] FREUND, T. - LITTLE, M. *OASIS Web Services Business Activity Version 1.2* [online]. 2. února, 2009[cit. 2012-05-23]. Dostupné z WWW: <http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os/wstx-wsba-1.2-spec-os.html>.

[11] HANSEN, M. D. *SOA Using Java Web Services*. Prentice Hall, 2007. 574 s. ISBN 0130449687.

[12] JENDROCK, E. - EVANS, I. *The Java EE 5 Tutorial* [online]. 2011 [cit. 2012-05-23]. Dostupné z WWW: <http://docs.oracle.com/javaee/5/tutorial/doc/>.

[13] LITTLE, M. - MARON, J. - PAVLIK, G. *Java Transaction Processing: Design and Implementation*. Prentice Hall, 2004. 402 s. ISBN 013035290X.

[14] LITTLE, M. - WILKINSON, A. *OASIS Web Services Atomic Transaction Version 1.2* [online]. 2. února, 2009 [cit. 2012-05-23]. Dostupné z WWW: <http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html>.

[15] MARÉCHAUX, J.-L. *Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus* [online]. 2012 [cit. 2012-05-23]. Dostupné z WWW: <http://www.ibm.com/developerworks/library/ws-soa-eda-esb/>.

[16] MITRA, N. - LAFON, Y. *SOAP Version 1.2 Part 0: Primer (Second Edition)* [online]. 2007 [cit. 2012-05-23]. Dostupné z WWW: <http://www.w3.org/TR/soap/>.

[17] RAGGETT, D. - HORSE, A. L. - JACOBS, I. *HTML 4.01 Specification* [online]. 1999 [cit. 2012-05-23]. Dostupné z WWW:<http://www.w3.org/TR/html4/>.

[18] SAGE, D. L. - MISON, D. *Jboss ESB Programmers Guide* [online]. 2012 [cit. 2012-05-23]. Dostupné z WWW: <http://docs.jboss.org/jbossesb/docs/4.11/manuals/html/Programmers_Guide/index.html>.

[19] SERAIN, D. *Middleware and Enterprise Application Integration: The Architecture of E-Business Solutions*. Springer, 2002. 288 s. ISBN 185233570X.