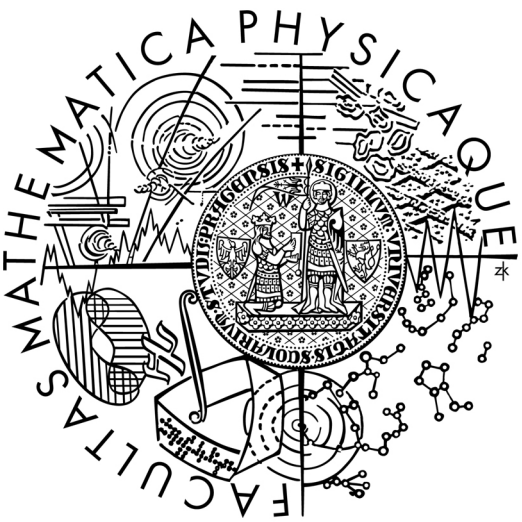
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Pavel Kadlec

Distribuované zpracování business procesů v prostředí podnikové sběrnice služeb

Katedra softwarového inženýrství
Vedoucí diplomové práce: *Prof. RNDr. Jaroslav Král, DrSc.*
Studijní program: *Informatika, softwarové systémy*

Především děkuji Prof. RNDr. Jaroslavu Královi, DrSc. za vedení diplomové práce, za jeho cenné rady a připomínky. Děkuji Pavlu Köhlerovi a Vladimíru Klementovi, spolumajitelům společnosti Flynet, za poskytnutí přístupu k systému Club a za umožnění využívání podnikových serverů. Děkuji všem zaměstnancům společnosti Flynet za spolupráci a konzultace při vývoji. A velmi děkuji mým blízkým za podporu, pochopení a trpělivost.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne *8.12.2008* *Pavel Kadlec*

Obsah

Název práce: *Distribuované zpracování business procesů v prostředí podnikové sběrnice služeb*
Autor: *Pavel Kadlec*
Katedra (ústav): *Katedra softwarového inženýrství*
Vedoucí diplomové práce: *Prof. RNDr. Jaroslav Král, DrSc.*
e-mail vedoucího: *Jaroslav.Kral@mff.cuni.cz*
Abstrakt: *Práce se zabývá návrhem frameworku, který umožní distribuované zpracování vnitropodnikových dávkových business procesů. Framework by měl především zrychlit zpracování business procesů a umožnit jejich monitoring a řízení.*

Autor práce se rozhodl použít podnikovou sběrnici služeb s cílem vytvořit výkonný distribuovaný framework, jehož používání bude příjemné pro aplikační programátory. Business procesy jsou dekomponovány na části – služby, které jsou nasazovány na jednotlivé uzly podnikové sběrnice za účelem dosažení distribuovaného zpracování. Služby mohou být nasazovány na jednotlivé uzly v mnoha instancích, aby bylo dosaženo také paralelního zpracování. Služby jsou koordinovány procesním manažerem podle předem nadefinovaného plánu. To znamená, že aplikační programátor se stará pouze o naprogramování business logiky jednotlivých služeb.

Motivací práce bylo zrychlit reálný business proces, který zpracovává velké množství dat. Zrychlení bylo dosaženo postupným aplikováním databázových optimalizací a následným začleněním několika počítačů do procesu zpracování prostřednictvím podnikové sběrnice služeb.

Základem distribuovaného frameworku je podniková sběrnice JBossESB, procesní manažer jBPM a distribuované transakce koordinované protokoly rodiny WS-Transaction.
Klíčová slova: *distribuované zpracování, podniková sběrnice služeb, business proces*

Title: *Distributed enactment of business processes in ESB framework*
Author: *Pavel Kadlec*
Department: *Department of Software Engineering*
Supervisor: *Prof. RNDr. Jaroslav Král, DrSc.*
Supervisor's e-mail address: *Jaroslav.Kral@mff.cuni.cz*
Abstract: *Thesis deals with design of framework that allows distributed processing of intradepartmental batch business processes. Framework should speed up the processing of business processes and enable their monitoring and management.*

Author of the thesis decided to use enterprise service bus in order to create a powerful distributed framework which will be easy to use by application programmers. Business processes are divided into parts – services that are deployed on individual nodes of enterprise service bus in order to achieve distributed processing. Services may be deployed on individual nodes in many instances in order to achieve parallel processing as well. Services are coordinated by process manager according to the plan. This means that the application programmer cares only about creating the business logic of individual services.

Motivation was to accelerate the real business process which handles a large amount of data. Acceleration was achieved by applying database optimizations and subsequent incorporation of several computers via the enterprise service bus.

Framework is based on enterprise service bus JBossESB, process manager jBPM and distributed transactions coordinated by WS-Transaction family protocol.
Keywords: *distributed processing, enterprise service bus, business process*

1 Úvod

Podniky v dnešní době spravují rozsáhlé množství dat a s těmito daty čas od času dávkově pracují. Například poskytují služby velkému počtu zákazníků a měsíčně tyto služby zpoplatňují. Takovýchto procesů může být v jednom období (například na konci měsíce) mnoho. Vzniká tak pro podnik velmi hektické období, ve kterém musí všechny tyto procesy stihnout. Mnohdy není možné provádět tyto procesy současně například kvůli hardwarovým omezením nebo z toho důvodu, že procesy musí následovat po sobě. Nevykonání všech procesů by mohlo vést k sankcím od úřadů, dalším peněžním ztrátám a v neposlední řadě k nespokojenosti klientů služeb.

Nezřídka i velké společnosti pro tyto výpočty používají jediný počítač. Ačkoli se často jedná o superpočítač, výpočty trvají až desítky hodin a naprosto vyčerpají výkon a hardwarové prostředky systému. Nasazení více počítačů střední kategorie by mohlo výrazně zrychlit zpracování těchto procesů. Řešení postavené na několika počítačích střední kategorie navíc za jistých okolností dokáže flexibilně a ekonomicky reagovat na rostoucí objem spravovaných dat. Naproti tomu upgrade superpočítače může být z ekonomických důvodů neakceptovatelný a mnohdy nevede k výraznému zvýšení výkonu. Celkově vzato je řešení postavené na několika počítačích střední kategorie výrazně lacinější než to řešení, které stojí na jednom superpočítači.

Infrastruktura, která je popsaná v této práci, je reakcí na problémy, které řešili zaměstnanci společnosti Flynet s.r.o. Společnost Flynet vyvíjí a administruje pro svého zahraničního partnera systém provozovaný jedním z největších evropských koncernů. Tento systém obsahuje stovky gigabajtů dat.

Společnost Flynet systém začala vyvíjet na J2EE aplikačním serveru JBoss. Aplikační server poskytuje mnoho služeb, které ulehčují aplikačním programátorům práci. V ideálním případě se aplikační programátor stará pouze o business logiku, protože aplikační server „vše ostatní“ řeší za něj. J2EE nabízí velmi intuitivní používání transakcí, intuitivní objektový přístup k databázi apod. V J2EE ovšem chybí možnost jednoduše vyvinout distribuovanou aplikaci používající asynchronní middleware – aplikační programátor se musí vždy starat o technické detaily komunikace. To byl jeden z důvodů, proč se zaměstnanci společnosti Flynet rozhodli navrhnout centralizovaný systém, který se ovšem potýká s výše uvedenými problémy. Autor práce se rozhodl použít podnikovou sběrnici služeb s cílem vytvořit výkonný distribuovaný framework, jehož používání bude příjemné pro aplikační programátory a jehož možnosti budou o mnoho větší než možnosti nativních prostředků J2EE. Pokusy ukázaly, že tímto způsobem lze výpočty podstatně urychlit a zefektivnit.

2 Záměr práce

Před tím, než bude uveden záměr práce, je nutné definovat základní pojmy tak, aby byl záměr jasný a začleněný do širšího kontextu.

Distribuované zpracování programu je takové zpracování, ve kterém je program rozdělený na několik menších, méně náročných úloh, které se zpracovávají na více počítačích propojených sítí. Lze ho využít jen u programů, jejichž algoritmus lze paralelizovat – převést na paralelní verzi, kdy vzájemně nezávislé části programu běží současně [1].

Business proces (BP) je kolekce vzájemně propojených aktivit. Aktivity zpracovávají vstup a produkují výstup, z kterého má zákazník prospěch [2].

Autor práce se zabývá speciálním typem BP – **dávkovými vnitropodnikovými** business procesy. Zákazníkem je v tomto případě sám podnik. Podnik využívá business procesy pro plnění svých podnikových cílů, povinností, plánů, vytváření statistik apod. Příkladem takového procesu je zpracování faktur nebo přičítání úroků k účtům. Business proces hraje roli programu v definici distribuovaného zpracování.

2.1 Cíle práce

1. Diskuse možností distribuovaného zpracování BP v prostředí J2EE [3].
2. Návrh a vytvoření infrastruktury (frameworku), která umožní distribuované zpracování BP v prostředí J2EE aplikačního serveru JBoss. Cílem návrhu je především výkonnost, rozšiřitelnost a škálovatelnost. Infrastruktura bude nasazena v prostředí lokální sítě. Používání infrastruktury by mělo být z pohledu aplikačního programátora transparentní.
3. Ověření navržené infrastruktury na konkrétním BP z reálného světa.

3 Reálné prostředí podniku, problémy a hlavní cíle

3.1 Kontext implementované úlohy

V dnešní době se zmenšují rozdíly mezi kvalitou produktů automobilových značek. Výrobci se snaží přilákat nové zákazníky a udržet si stávající klienty nadstandardními službami. Nabízejí zákazníkům speciální služby, akce a zavádí různé věrnostní kluby. Jeden z největších evropských koncernů[1] se rozhodl založit věrnostní kluby pro své automobilové značky. Každá automobilová značka spravuje svůj vlastní klub. Říkáme, že automobilová značka je mandantem klubu. Členové klubů jsou cenově zvýhodňováni u partnerů automobilek, tj. v autorizovaných servisech, ve vybraných čerpacích stanicích, v hotelech apod.

Člen klubu vlastní klubovou kartu, na kterou získává body za své nákupy, opravy apod. Za klubové body si pak může podle daného přepočtu na peníze u jakéhokoli partnera cokoli koupit. Mandant klubu posléze tuto transakci partnerovi proplatí. V podstatě se jedná o klasickou bezhotovostní platbu.

Členové klubu rovněž pravidelně dostávají magazín, ve kterém jsou informováni o nových produktech automobilky, o službách jednotlivých partnerů apod.

Každá ze zúčastněných stran má z klubu prospěch:

- Mandant přiláká zákazníky na novou službu a udrží si staré. Prodá více svých produktů, protože do bodového programu řadí pouze originální díly a doplňky.
- Partner mandanta je propagován v magazínu renovované automobilky a je upřednostňován zákazníky pro možnost získání bodů či placení klubovými body.
- Člen klubu ušetří peníze.

Společnost Flynet vyvinula informační systém za účelem spravování výše zmíněných věrnostních klubů. Společnost Flynet systém i administruje. Systém běží na J2EE aplikačním serveru JBoss. V následujícím textu se tento systém nazývá Club.

3.2 Problémy zpracování dat v systému Club a hlavní cíle práce

Na konci každého měsíce musí administrátor BP (administrátor business procesů) spustit posloupnost velmi důležitých business procesů. Vzhledem k rostoucímu množství dat (především rostoucímu počtu uživatelů), systém začal mít neúměrně vysoké nároky na výpočetní kapacitu. V systému Club docházelo k následujícím problémům:

- **Časově náročné business procesy**: Některé business procesy trvaly velmi dlouho, někdy až 20 hodin. To se stalo neúnosné, protože business procesy mají striktní termíny, do kterých musí být skončeny. Nestihnutí zpracování BP do daného termínu může znamenat sankce od úřadů, peněžní ztráty a v neposlední řadě nespokojenost zákazníků.

1 Licenční podmínky a zabezpečení neumožňují zveřejnění jména koncernu

- **Hardwarové nároky**: Některé BP vyžadovaly přes 8 GB paměti. Takto vysoké nároky na paměť byly jednou z hlavních příčin toho, že BP byly příliš pomalé.
- **Nemožnost řídit business proces**: Administrátor BP dohlíží na jeho průběh. Administrátor BP mohl spustit zpracování, ale již neměl absolutně žádnou kontrolu nad dalším průběhem procesu. Mnohdy vzniklá situace vyžadovala předčasné ukončení BP, ale systém Club tuto možnost nenabízel.

Na základě výše uvedených problémů vznikla idea překlopit business logiku systému Club na infrastrukturu, která by výše uvedené problémy eliminovala. Byly stanoveny následující podcíle:

1. **Vytvoření škálovatelné distribuované infrastruktury**: Infrastruktura musí umožnit distribuované zpracování BP. Z pohledu aplikačního programátora musí být začlenění jednotlivých počítačů do procesu zpracování transparentní. Je důležité zvolit vhodný middleware pro meziprocesovou komunikaci.
2. **Maximální využití paralelity algoritmu BP**: Podmínkou pro to, aby bylo možno zpracovávat BP ve více paralelních větvích, je, že BP musí jít dekomponovat na více nezávislých částí, které lze zpracovávat současně.
3. **Zrychlení zpracování BP a zefektivnění využívání hardwarových prostředků (především procesorového výkonu a paměti)**: Je nutné najít největší nehospodárnost ve zpracování BP. Vysoké nároky na paměť poukazují na to, že v systému Club není něco v pořádku.
   Distribuovaným zpracováním BP se pokusit co nejvíce zvýšit výkon. S distribuovaným zpracováním jde ruku v ruce snížení paměťové náročnosti na jeden počítač.
4. **Kontrola a řízení BP**: V některých situacích se hodí BP pozastavit nebo úplně přerušit. Administrátor BP by měl mít prostřednictvím administrátorského rozhraní možnost výpočet řídit. Zejména jsou vhodné operace předčasného ukončení, pozastavení (suspend) a opětovného rozběhnutí (resume) BP.
5. **Vykonávání BP v rámci jedné distribuované transakce**: Části BP mohou probíhat současně a na různých počítačích. Je nutné, aby tyto části BP vykonávaly svou práci v rámci kontextu společné distribuované transakce. V případě chyby je nutné BP co **nejdříve** ukončit a vrátit změny, které BP způsobil, zpět.
6. **Oddělení komunikační logiky a business logiky**: V původní implementaci systému Club se aplikační programátor musí starat o veškerou komunikaci, posílání zpráv, práci s vlákny. Komunikační logika je v původní implementaci zapouzdřena do business logiky.

   Je důležité, aby komunikační logika a business logika od sebe byly odděleny. To způsobí zpřehlednění kódu pro budoucí údržbu. Povede to k zrychlení vývoje business logiky.

   Jak již bylo řečeno, distribuované zpracování lze využít jen u business procesů, jejichž algoritmus lze paralelizovat – převést na paralelní verzi, kdy vzájemně nezávislé části business procesu běží současně. V ideálním případě by se

aplikační programátor měl starat pouze o naprogramování jednotlivých „částí“ business procesu. Samotné spojení těchto částí může mít na starost někdo jiný, např. architekt business procesů (architekt BP). Spojení částí BP by již nebylo realizováno ručním kódováním, ale grafickým navrhováním BP pomocí nějakého CASE nástroje. Architekt v CASE nástroji navrhne plán BP – v řeči Unified Modeling Language (UML) vytvoří Activity diagram [4]. Tento plán „nahraje“ do vytvořené infrastruktury. Tím je BP připraven k použití.

7. **Maximální znovupoužití implementace stávajících BP**: Původní BP jsou homogenní v tom smyslu, že jsou celé napsány v EJB 3.0 [5]:
   - Business logika je zapouzdřena do stateless session bean (SLSB) [6] a message driven bean (MDB) [7].
   - Přístup k databázovým tabulkám je zajištěn pomocí stateless session bean. EJB, které slouží pro zpřístupnění dat z databáze, se obecně nazývají data access objects (DAO) [8].
   - Databázové tabulky jsou objektově reprezentovány pomocí entity bean [9]. Těmto EJB se obecně říká data transfer objects (DTO) [10].

   Znovupoužití DAO bean a DTO bean je vhodné a výhodné. Potíže jsou se znovupoužitím business logiky, ve které je zapouzdřena implementace algoritmu BP.
8. **Postupný přechod ze starého systému na novou infrastrukturu**: Není možné přejít ze dne na den na novou infrastrukturu. Ačkoli je původní řešení v mnoha ohledech nedostačující, je otestované provozem a funkční. Původní řešení je stále nasazeno na produkčním systému. Postupně se BP, u kterých je to vhodné, budou překlápět do nové infrastruktury. Je tedy nutná koexistence stávajícího a nového systému.
9. **Transparentnost spojení původního a nového systému z pohledu administrátora BP**: Původní systém Club obsahuje rozhraní pro administrátora BP. Bylo by vhodné, aby vytvořená infrastruktura používala stejné rozhraní. Používání dvou rozhraní je značně nešikovné a činí administraci zbytečně nepohodlnou a nepřehlednou.
10. **Monitoring infrastruktury**: Informace získané sledováním chování business procesu, míry komunikace mezi jednotlivými počítači apod. přispěje k nalezení úzkých míst ve zpracování business procesu a pomůže ho optimalizovat.

4 Postup řešení

1. **Vytvoření infrastruktury**: Nejprve je potřeba vytvořit takovou infrastrukturu, která by splňovala cíle vytyčené v kapitole 3.2.

   Vytvoření infrastruktury je složeno z následujících kroků:

   1. Volba middlewaru.
   2. Volba nástavby nad middlewarem, která transparentně poskytne aplikačnímu programátorovi služby middlewaru.
   3. Volba a integrace distribuovaných transakcí do zvolené nástavby.

2. **Integrace business procesu z reálného světa**: Motivací pro vznik infrastruktury byl původně BP trvající 20 hodin a mající obrovské nároky na paměť. Tento BP je ideální kandidát na otestování infrastruktury.

3. **Analýza zpracování BP ve vytvořené infrastruktuře**: Aby bylo možné kvantifikovat přínos infrastruktury, je nutné zjistit, jak se infrastruktura chová a jaké přináší výsledky.

5 Volba middlewaru

Máme na výběr mezi asynchronní komunikací a synchronní komunikací. Vzhledem k vytyčeným cílům je nejvhodnější použít posílání zpráv (messaging) [11] jako komunikačního prostředku. Hohpe a Woolf uvádějí následující výhody messaging [12]:

- **Asynchronní komunikace**: Messaging umožňuje tzv. *send-and-forget* přístup ke komunikaci. Poté co volající aplikace odešle zprávu, nemusí čekat na odpověď a může okamžitě dál pokračovat ve své činnosti.

- **Proměnné časování**: Při použití synchronní komunikace musí volající čekat na odpověď od příjemce, aby mohl pokračovat dál v práci. To znamená, že volající může být pouze tak rychlý jako příjemce. Asynchronní komunikace dovoluje odesílateli vysílat požadavky svým vlastním tempem a stejně tak dovoluje příjemci svým vlastním tempem požadavky přijímat. To umožňuje odesílateli i příjemci pracovat na maximální možné frekvenci (pokud tedy příjemce má neustále zprávy ke zpracování).

- **Předcházení zahlcení příjemce**: Problémem synchronní komunikace může být zahlcení příjemce přílišným množstvím volání najednou. To může způsobit utlumení až pád příjemce. Messaging toto řeší ukládáním požadavků do fronty. Příjemce je poté zpracovává s libovolnou frekvencí. Tím se předchází zahlcení příjemce. Volající není příjemcem blokován, protože komunikace je asynchronní.

- **Škálovatelnost**: Messaging umožňuje měnit počet konzumentů zpráv (messaging služeb) a rozmisťovat je na různé uzly systému.

- **Transakční komunikace**: Speciálně Java Message Service [13] umožňuje zprávu poslat v rámci aktuální transakce a stejně tak přijetí zprávy může být součástí transakce, ve které se obsah zprávy zpracovává. Zpráva je z fronty odstraněna až momentě, kdy konzument zprávu úspěšně zpracuje. Pokud se použije tzv. persistentní fronta, tak nedojde ke ztrátě zpracovávané zprávy ani v případě pádu konzumenta.

- **Přeposlání zprávy**: S pomocí JCA (J2EE Connector Architecture [14]) lze docílit toho, že pokud se zpracování obsahu zprávy z nějakého důvodu nezdaří, zpráva je opětovně předána konzumentovi ke zpracování. Pokud se vyčerpá počet pokusů na zpracování, je zpráva poslána do určené fronty jako nezpracovaná zpráva.

- **Služby navíc**: Messaging systém poskytuje konzumentům a producentům redundantní zdroje k zajištění vysoké dostupnosti, vyvažování zátěže, kvalitu služeb (Quality of service [15]), přeposlání zprávy v případě výpadku síťového spojení aj.

Posílání zpráv ovšem není všelék. Messaging se potýká s následujícími problémy [12]:

- **Složitější programový model**: Vývojáři v případě messaging pracují s tzv.

event-driven programovacím modelem [16]. Již to není model, kde by jedna procedura volala jinou proceduru. Systém se sestává z *event-hadlerů*. Takový systém je složitější, hůře programovatelný a obtížnější na ladění.

- **Pořadí zpráv**: Zprávy mohou být přijaty v jiném pořadí, než v jakém byly odeslány.
- **Bezestavoví konzumenti zpráv**: V middlewaru, který je založený na posílání zpráv, se složitěji simuluje dlouhotrvající komunikace. Ve své podstatě je konzument zpráv bezestavový (stateless) – v momentě, kdy konzument přijatou zprávu zpracuje, je veškerý kontext této zprávy ztracen a konzument ho nemůže použít při zpracování další zprávy. Kontext delší komunikace (např. identifikátor zákazníka, jeho nákupní košík apod.) musí být přenášen v nitru zprávy. Navíc producent zpráv si nemůže být jist, že všechny odeslané zprávy přijme stejný konzument. Alternativa k přenášení kontextu uvnitř zpráv je vyčlenění služby, která kontexty spravuje a na vyžádání poskytne.
- **Výkon**: Posílání zpráv vyžaduje dodatečné zdroje. Jde především o zabalení dat do zprávy, poslání zprávy, přijmutí zprávy, vybalení obsahu a pak až zpracování. Pro zajištění vysoké spolehlivosti se fronta nebo topic nakonfiguruje jako persistentní. Persistentní fronty jsou oproti in-memory frontám o mnoho pomalejší.
- **Synchronní scénáře komunikace**: Některé aplikace vyžadují synchronní komunikaci. Messaging systém musí mít prostředky jak tuto synchronní komunikaci nasimulovat.

Volba middlewaru, který využívá posílání zpráv, je základním kamenem pro vytvoření škálovatelné, výkonné a spolehlivé infrastruktury.

5.1 Řízení toku zpráv

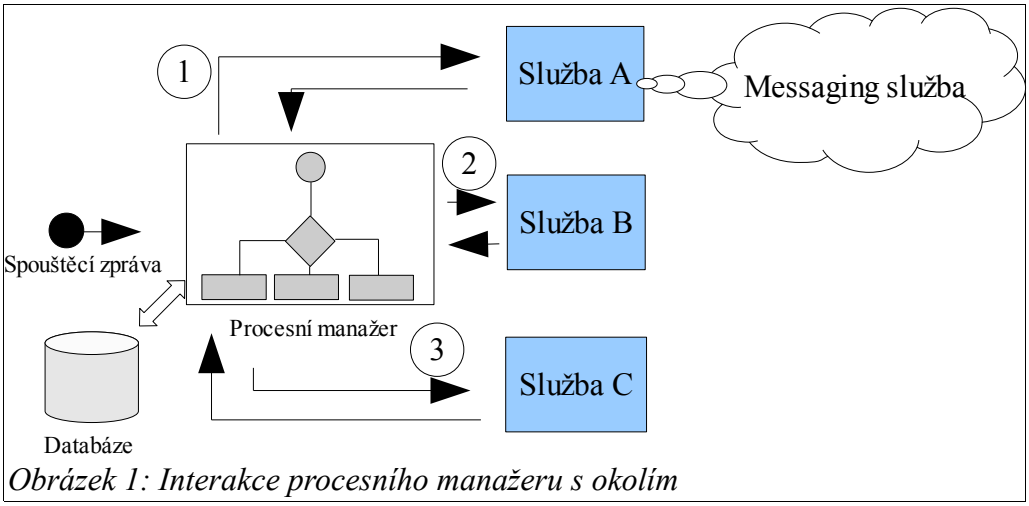
Rozhodnutí vydat se cestou messaging s sebou nese otázku, jak řídit tok zpráv. Hohpe a Woolf uvádějí vzory, kterými lze komunikaci řídit. Základními vzory pro řízení toku zpráv jsou [12]:

- **Message Router**: Zpráva je podle svého obsahu posílána na různé destinace. Router na obsah zprávy aplikuje podmínky a podle výsledku vybere destinaci, které zprávu doručí. Router je nakonfigurován staticky (množina podmínek a množina destinací).
- **Splitter**: Z jedné zprávy obsahující seznam položek vytváří n-tici zpráv obsahujících jednotlivé položky.
- **Aggregator**: Aggregator agreguje n-tici zpráv do jedné zprávy. N-tice vstupních zpráv je nějakým korelačním identifikátorem svázaná. Aggregator se používá v páru se splitterem.

Tyto tři vzory (patterns) poskytují dostatečnou flexibilitu, ale jejich použití je nevýhodné v tom smyslu, že routing logika je rozmístěna ve více komponentách (třeba i na různých uzlech).

Můžeme získat flexibilitu a zároveň mít centrální řídící komponentu, pokud po každém vyvolání nějaké „messaging služby“ vrátíme řízení do nějaké centrální komponenty. Centrální komponenta může určit další „messaging službu“ a vyvolat ji. Tímto způsobem má centrální komponenta kontrolu nad celým procesem. Hohpe a Woolf uvádějí, že touto centrální komponentou může být tzv. procesní manažer (anglicky *Process Manager)* [12].

Procesní manažer umí určit další krok na základě průběžných výsledků a pozice v předem nadefinovaném plánu procesu. Pokud je centrální komponenta vybavená nějakou formou paměti, nemusí přenášet celý kontext procesu v jednotlivých zprávách. Centrální komponenta ze své „paměti“ zjistí, jaký má provést následující krok.



Obrázek 1: Interakce procesního manažeru s okolím

Procesní manažer a jednotlivé „služby“ dohromady tvoří tzv. *hub-and-spoke* architekturu. Spouštěcí zpráva inicializuje procesní manažer. Na základě plánu procesu pošle procesní manažer zprávu (1) službě A. Poté co služba A dokončí svou práci, pošle odpověď zpět procesnímu manažeru. Procesní manažer poté určí další službu a pošle jí zprávu (2). Komunikace tedy probíhá přes centrální „hub“, proto termín hub-and-spoke. Hlavními funkcemi procesního manažeru jsou:

- **Řízení instance procesu**: Procesní manažer se stará o řízení instance procesu. Musí být schopen řídit více instancí (stejných, nebo různých procesů) najednou.
- **Spravování kontextu procesu**: Uchovávání informací (např. obsah nákupního košíku), které sice nejsou relevantní pro aktuální krok, ale mohou být potřebné pro kroky následující.
- **Administrace procesů**: Administrátor musí mít možnost procesy spouštět, pozastavovat a kontrolovat jejich stav.

Každá architektura s centrálním prvkem je závislá na životě tohoto centrálního prvku (tzv. *single point of failure*). Z tohoto důvodu by měl procesní manažer dovolovat uchovávat kontext procesní instance v databázi. Pro zvýšení robustnosti a výkonnosti je vhodné mít puštěno několik paralelních procesních manažerů napojených na jednu databázi. Nevýhoda tohoto přístupu je, že se musí persistentně ukládat kontext procesní instance po každém kroku manažeru, což je časově náročné [12].

6 Volba nástavby nad middlewarem založeným na posílání zpráv - diskuse možností distribuovaného zpracování BP

V předchozí kapitole autor zvolil zprávy jako vhodný komunikační prostředek ve vytvářeném frameworku pro distribuované zpracování business procesů. V této kapitole jsou analyzovány možnosti jak dosáhnout distribuovaného zpracování v prostředí J2EE. Zaměřuje se na řešení, které využívá pouze J2EE prostředků a na řešení, které využívá vlastností podnikové sběrnice služeb.

6.1 Distribuovaného zpracování BP v prostředí J2EE

Systém Club běží na platformě J2EE. Je přirozené se nejprve podívat na možnosti, které tato platforma poskytuje pro distribuované zpracování. V této kapitole bude popsána infrastruktura, která bude používat pouze prostředků J2EE. Systém Club je implementován tímto způsobem.

Základním kamenem této infrastruktury jsou Message Driven Beany (MDB) [7]. MDB je enterprise beana [17], která umožňuje J2EE aplikaci zpracovávat zprávy asynchronně. MDB běží v J2EE kontejneru, který poskytuje skvělou podporu pro business logiku (transakce, persistentní vrstva atd.). MDB se chová se jako JMS listener, který je podobný event-listeneru až na to, že přijímá zprávy místo událostí. Zprávy mohou být poslány libovolnou J2EE komponentou – aplikačním klientem, jinou enterprise beanou, webovou komponentou, JMS aplikací nebo systémem, který nepoužívá J2EE technologii.

6.1.1 Postup při implementaci distribuovaného business procesu

Aby mělo smysl zpracovávat business proces na více počítačích, je zapotřebí, aby v BP existovaly části, které lze provádět paralelně.

Příkladem je business proces, který:

1. načte identifikátory bankovních účtů z databáze,
2. připočítá úroky,
3. uloží aktualizované stavy účty do databáze.

Implementace v prostředí J2EE se skládá ze dvou bean:

- Stateless session beana **ReadAllAccountIdsBean** provede následující operace:
  1. načte z databáze identifikátory bankovních účtů,
  2. každý identifikátor zabalí do zprávy a každou zprávu pošle do fronty A.
- Message driven beana **AddInterestAndPersistAccountMDB** poslouchá na frontě A. MDB je například nakonfigurovaná tak, že může zpracovávat až 10 zpráv současně. MDB provede následující operace:

1. vyzvedne z fronty zprávu,
2. načte účet s daným identifikátorem,
3. připočítá úrok,
4. aktualizovaný účet uloží do databáze.

Získali jsme paralelního zpracování jednotlivých bankovních účtů. Tímto způsobem se standardně dosahuje paralelního zpracování v J2EE. Za předpokladu, že je fronta distribuovaná, tak jednotlivé message driven beany **AddInterestAndPersistAccountMDB** mohou být vystaveny na více počítačích. JMS provider zajistí vyvažování zátěže podle nakonfigurované politiky.

Výhody řešení:

- Řešení je velmi jednoduché. K jeho implementaci stačí základní znalost J2EE.
- Při použití Java Transaction Service [18] můžeme práci v jednotlivých MDB provádět v rámci jedné distribuované transakce.

Nevýhody:

- Aplikační programátoři jsou nuceni **explicitně** řídit tok zpráv. O to by se aplikační programátor neměl starat. Tok zpráv je nastálo zapouzdřen do jednotlivých message driven bean.
- Monitorování provozu poskytované aplikačním serverem je nedostatečné. Nemáme přehled např. o době zpracování zpráv, o počtu úspěšně/neúspěšně zpracovaných zpráv.
- Nikdo neví, kdy business proces skončí. MDB **AddInterestAndPersistAccountMDB** sice může posílat odpověď nějaké singleton message driven beaně, která bude počítat, zda-li všechny odpovědi dorazily, ale tím se jednoduchost řešení vytrácí.
- Musíme se starat o obsluhu výjimek. Aplikační programátor musí řešit, kam poslat chybovou zprávu. Jaká komponenta celý výpočet v případě chyby přeruší?
- Aplikační server neposkytuje možnost, jak takovýto proces řídit standardními prostředky. Při této implementaci nelze proces pozastavit, opětovně rozběhnout ani předčasně přerušit.
- Pro implementaci paralelního zpracování pomocí posílání zpráv je vhodné užít vzory, které uvádějí Hohpe a Woolf [12]. Především vzory Splitter a Aggregator uvedené v kapitole 5.1 jsou nezbytné. Tyto vzory by bylo nutné naprogramovat takovým způsobem, aby byly znovupoužitelné i v plánech jiných business procesů.

Jistě by šlo spoustu nevýhod odstranit vlastním více či méně komplikovaným řešením. Je otázka, do jaké míry by se povedlo vytvořit znovupoužitelné řešení a jestli by řešení bylo vůbec použitelné. Autor diplomové práce se podíval prostředky, které se

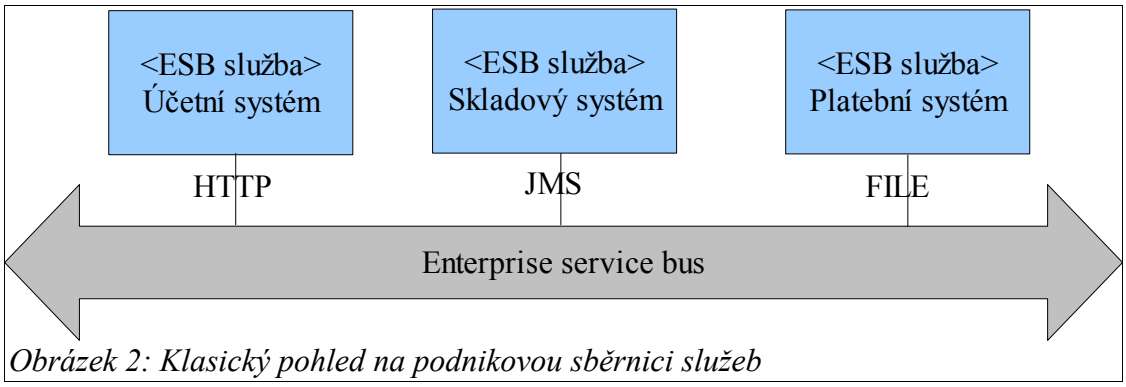
používají v jiných (avšak ne úplně vzdálených) oblastech informatiky a pokusil se pomocí těchto prostředků vytvořit framework, který by výše uvedené nevýhody eliminoval.

6.1.2 Distribuované zpracování BP v prostředí podnikové sběrnice služeb

Podniková sběrnice služeb [19] (ESB – Enterprise Service Bus) poskytuje abstraktní vrstvu nad implementací EMS [20] (Enterprise Messaging systém), což umožňuje programátorům využít hodnoty posílání zpráv bez nutnosti psát vlastní kód. EMS je sada publikovaných standardů, které umožňují posílat sémanticky přesné zprávy mezi počítačovými systémy. Podniková sběrnice služeb podporuje více transportních protokolů jako např. posílání JMS [13] zpráv, SOAP [21] zprávy, komunikace skrz soubory, databázi apod.

Základními prvky ESB jsou služba, zpráva a sběrnice. ESB transparentně služby propojuje pomocí sběrnice. Komunikace mezi službami je zajištěna pomocí zpráv, jejichž formát je pro všechny transportní protokoly stejný. Každá služba si sama určuje, jakým podporovaným protokolem se ke sběrnici připojí.

Klasicky se podniková sběrnice služeb používá pro spojování heterogenních podnikových systémů (viz obrázek 2).



Obrázek 2: Klasický pohled na podnikovou sběrnici služeb

Mezi klíčové funkce ESB patří [19]:

- vyvolání služeb,
  - podpora synchronních a asynchronních transportních protokolů, vyhledávání služeb, vyvažování zátěže,
- směrování,
  - adresovatelnost, statické směrování, směrování na základě obsahu,
- zprostředkování,
  - adaptéry, překlad protokolů,
- orchestrace služeb,

    - vytváření business procesu (nebo jeho části) koordinací několika jednotlivých služeb,
- kvalita služeb,
    - bezpečnost, spolehlivý přenos, transakce,
- management,
    - monitorování, měření, administrátorské rozhraní.

Neexistuje standard, který by specifikoval, jak má ESB vypadat. Výše zmíněné funkce by ale měla poskytovat každá implementace podnikové sběrnice služeb. V současnosti existuje na trhu několik kvalitních implementací ESB. Mezi nejvýznamnější patří IBM Websphere Enterprise Service Bus, BEA Aqualogic Service Bus, Mule, ServiceMix, JBossESB, FUSE a mnoho dalších. Komerční ESB produkty jsou dostupné za nemalé licenční poplatky. V případě open source produktů tento problém odpadá. Podle InfoWorld Best Of Open Source Software Awards [22] je nejlepším open source ESB produktem za rok 2008 JBossESB [23].

Systém Club je nasazen na aplikačním serveru JBoss. Vzhledem ke snaze maximálně znovupoužít původní implementaci business procesů se autor rozhodl nasadit podnikovou sběrnici JBossESB, protože je s aplikačním serverem JBoss plně integrovaná.

Výhody a nevýhody použití podnikové sběrnice služeb JBossESB jsou sepsány v kapitole 7 poté, co budou představeny její funkce.

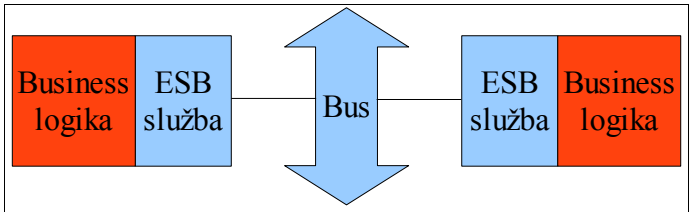
7 Podniková sběrnice JBossESB

Cílem této kapitoly je podrobněji představit hlavní funkce JBossESB [23] a ukázat, jak lze použít podnikovou sběrnici služeb pro distribuované zpracování business procesů.

7.1 Architektura podnikové sběrnice

7.1.1 ESB služba

V řeči knihy Enterprise Integration Patterns [12] je ESB služba message endpoint [24]. Message endpoint odděluje business logiku od messaging systému. ESB služba umí komunikovat jak s business logikou tak s podnikovou sběrnicí služeb. Díky službě není do business logiky zanášen jakýkoli kód týkající se komunikace v ESB. Všechen tento „komunikační“ kód je aplikačnímu programátorovi skryt.



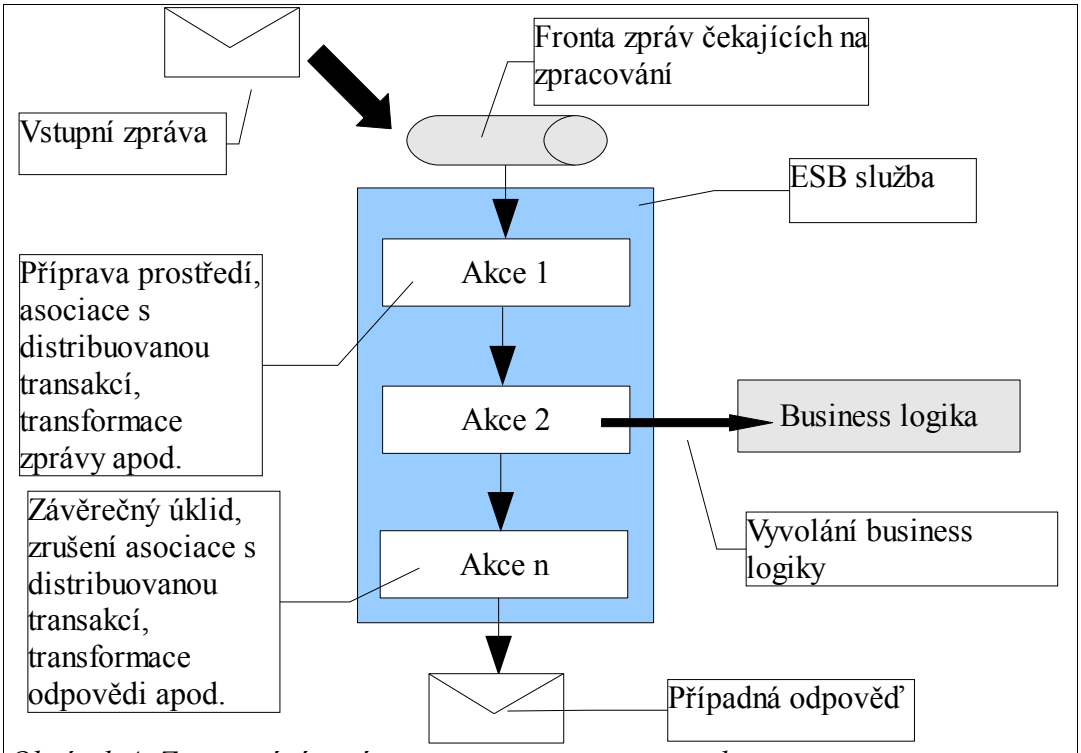
Obrázek 3: Odstínění business logiky od ESB

Služba je identifikovaná svým jménem. Klient musí znát jméno služby, aby jí mohl poslat zprávu.

Každá služba obsahuje tzv. action processing pipeline. Action processing pipeline je lineární sekvence uživatelských akcí. Výstup jedné akce je vstupem akce následující a o jejich vyvolání ve správném pořadí se stará jádro podnikové sběrnice služeb. Toto rozdělení služby na akce činí službu daleko přehlednější. Action processing pipeline se konfiguruje v XML souboru, takže služby lze z předem připravených akcí skládat bez nutnosti kompilace.

Action processing pipeline autor diplomové práce využívá zejména u těch služeb, jejichž práce se má vykonat v rámci distribuované transakce. První akce nastaví aktuální prostředí (vlákno) tak, aby bylo asociované s konkrétní distribuovanou transakcí. Druhá akce zavolá z tohoto nainicializovaného prostředí (vlákna) business logiku. Poslední akce vyčistí prostředí tzn. zruší asociaci prostředí (vlákna) s konkrétní distribuovanou transakcí. Action processing pipeline připomíná vzor (pattern) Pipes and Filters [25], který člení zpracování zprávy do nezávislých kroků.

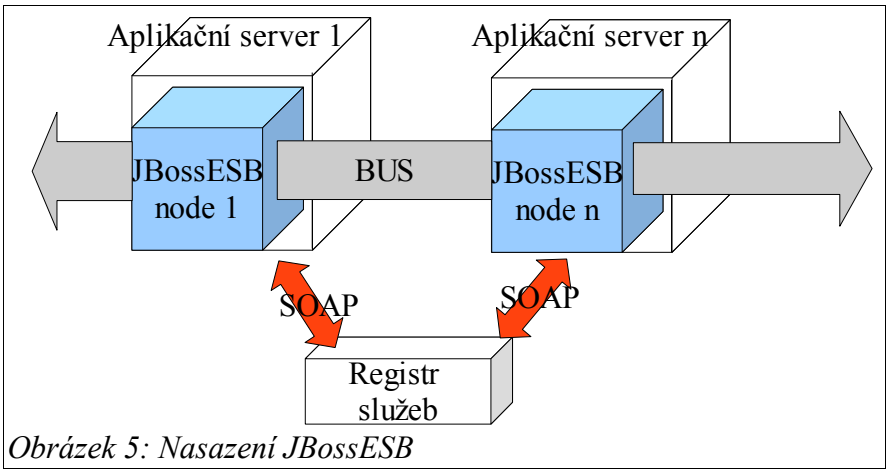
Každá služba je vyvolána zprávou. Zpráva zůstává ve frontě do té doby, dokud si ji action processing pipeline nevyzvedne. Action processing pipeline zprávu zpracuje a vyrobí odpověď. Na základě metadat ze vstupní zprávy je určena služba, které má být tato odpověď poslána. Pokud se během zpracování zprávy vyskytne chyba, je action processing pipeline okamžitě přerušena a na základě metadat ze vstupní zprávy je určena služba, které má být poslána chybová odpověď. Zpracování zprávy službou je zobrazeno na obrázku 4.



Obrázek 4: Zpracování zprávy v action processing pipeline

7.1.2 Distribuovaný framework

Distribuovaný framework se skládá z instancí produktů JBossESB nasazených na různé počítače. JBossESB potřebuje pro svůj běh aplikační server JBoss. Na jednom aplikačním serveru JBoss může být nasazena maximálně jedna instance JBossESB. Každá instance JBossESB tvoří tzv. uzel podnikové sběrnice. Jednotlivé instance (uzly) dohromady tvoří to, čemu se říká sběrnice (anglicky *bus*). Všechny instance sdílí společný registr služeb. Registr služeb je určen pro publikování a čtení metadat o službách. Obrázek 5 zobrazuje jednotlivé uzly ESB a společný registr služeb.



Obrázek 5: Nasazení JBossESB

Uzly ESB tvoří běhové prostředí pro ESB služby. Na každém uzlu může být nasazeno libovolné množství služeb.

Pro zvýšení výkonu může být stejná služba nasazena v podnikové sběrnici několikrát. Říkáme tomu, že je nasazeno několik instancí stejné služby. Instance služby mohou být vystaveny na libovolných uzlech podnikové sběrnice. Všechny instance mají stejné jméno služby. To znamená, že klient neví, které konkrétní instanci bude zpráva doručena (dále viz kapitola 7.2.2).

Každá instance služby má adresu (EPR – endpoint reference) uloženou v registru služeb. Instance stejné služby vystavené na různých uzlech ESB se liší svou adresou. Všechny instance stejné služby vystavené na stejném uzlu mají stejnou adresu.

7.2 Vlastnosti a funkce JBossESB využité pro zpracování business procesů

7.2.1 Orchestrace služeb

Orchestrace popisuje automatizované uspořádání, koordinaci a řízení služeb. Cílem orchestrace služeb je koordinace několika služeb za účelem vytvoření komplexního business procesu (nebo jeho části) [26].

Termín service orchestration se často užívá ve spojitosti s webovými službami. Pro orchestraci webových služeb se tradičně používá jazyk BPEL [27]. Avšak pokud koordinujeme služby podnikové sběrnice, BPEL použít nemůžeme. ESB služby nemají WSDL [28] rozhraní, ale mají rozhraní závislé na transportním protokolu, kterým se připojují ke sběrnici (JMS fronta, adresář, databázová tabulka aj.). Toto rozhraní je ale klientovi skryto. Klient používá pro vyvolání služby knihovnu podnikové sběrnice a o konkrétní transportní protokol se nezajímá. Klient službu identifikuje jejím jménem.

JBossESB je integrován s produktem jBPM [29]. jBPM je procesní manažer [30], který umí:

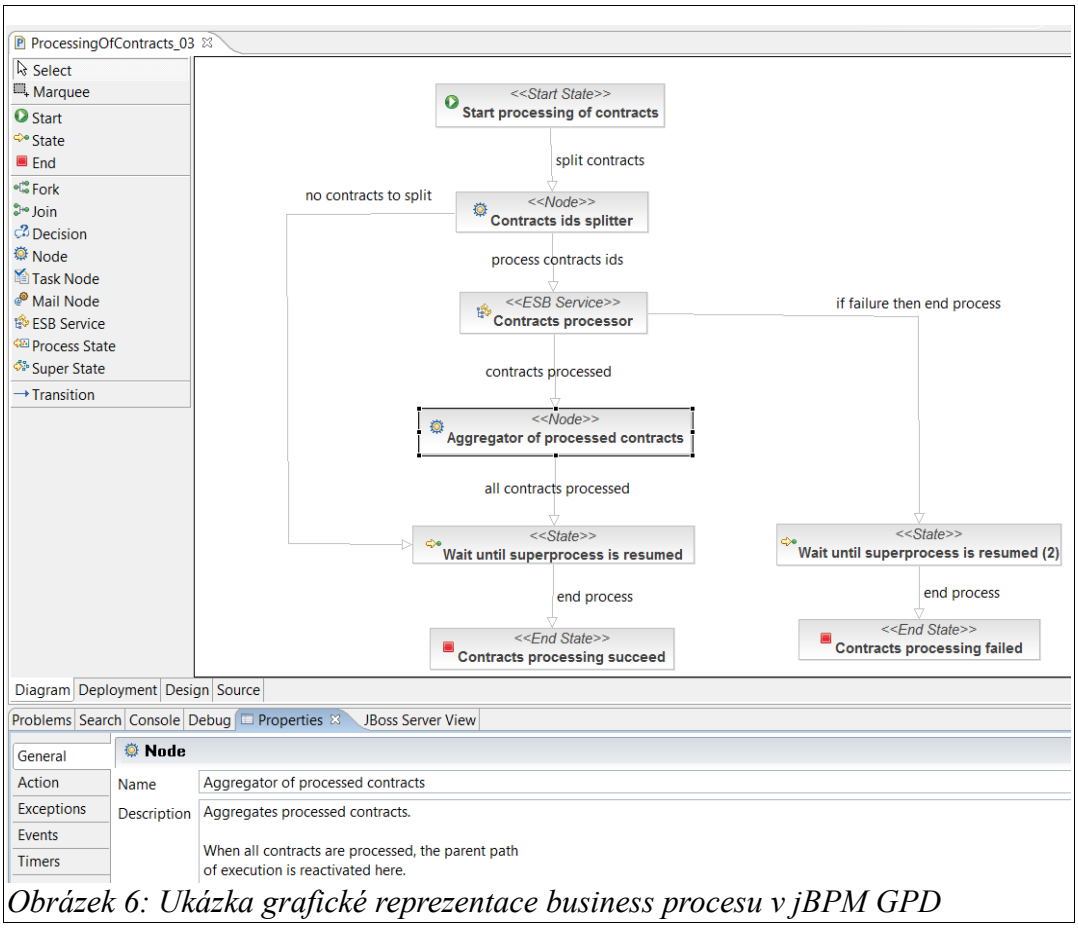
- koordinovat webové služby pomocí jazyka BPEL,
- koordinovat jakékoli Java komponenty pomocí jazyka jPDL.

Vývojáři JBossESB rozšířili schopnosti produktu jBPM tak, že navíc umí:

- koordinovat ESB služby pomocí jazyka jPDL.

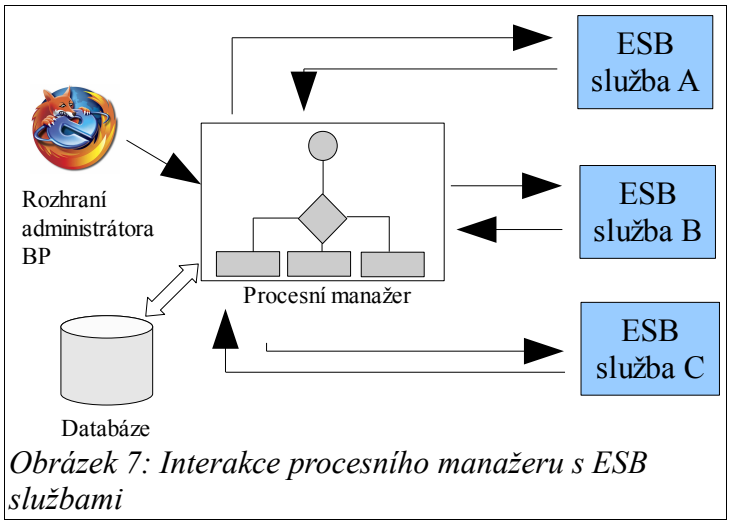
Jazyk jPDL [31] (jBPM Process Definition Language) je proprietární workflow [32] jazyk. Tvorba definice business procesu probíhá pomocí produktu jBPM GPD.

jBPM GPD [33] (Graphical Process Designer) je sada rozšíření pro vývojové prostředí Eclipse [34], která umožňuje vytvářet a editovat komplexní business procesy napsané v jazyce jPDL. Business procesy jsou v GPD reprezentovány orientovaným grafem. Na obrázku 6 je výřez z vývojového prostředí Eclipse. Na obrázku je příklad graficky reprezentovaného business procesu napsaného v jazyce jPDL. Na příkladu jsou koordinovány ESB služby. V definici se nachází několik uzlů, které řídí tok. Pro řízení toku se používá řada vzorů (patternů), které lze najít na webu Workflow Patterns [35].



Obrázek 6: Ukázka grafické reprezentace business procesu v jBPM GPD

Definice BP je uložená v databázi. Procesní manažer definici BP interpretuje. Procesním manažer je ovládán pomocí webového administrátorského rozhraní. Toto rozhraní umožňuje spouštět a předčasně ukončovat jednotlivé business procesy, pozastavovat je (suspend) a opětovně je rozběhnout (resume). Rozhraní rovněž informuje o době trvání BP, času startu a času ukončení BP. Na obrázku 7 je zobrazena interakce ESB služeb, procesního manažeru, databáze a administrátorského rozhraní.



Obrázek 7: Interakce procesního manažeru s ESB službami

Vlastnosti a funkce procesního manažeru:

- Interpretuje definici business procesu a koordinuje jednotlivé komponenty (včetně ESB služeb).
- Vyvolává jednotlivé komponenty (včetně ESB služeb) s definovanými parametry, jejichž hodnoty vznikají během běhu business procesu.
- Dovoluje současný běh více business procesů.
- Uchovává stav a kontext každého spuštěného business procesu v databázi.
- Kontext celého business procesu nemusí být přenášen v nitru zprávy. Ve zprávě se nacházejí jen ta data, která jsou nutná pro zpracování zprávy (odpovídá vzoru Claim Check [36]).

Aby se procesní manažer nestal úzkým místem infrastruktury, je na každý uzel podnikové sběrnice služeb nasazena jedna instance procesního manažeru. Všechny instance procesních manažerů přistupují ke stejné databázi. To znamená, že instance procesních manažerů sdílí kontexty běžících business procesů. Interpretace definice business procesu vypadá tak, že „kousek“ definice business procesu interpretuje jedna instance procesního manažeru, další kousek jiná (nebo stejná) instance procesního manažeru atd.

7.2.2 Vyvažování zátěže (load balancing)

V případě, že chceme poslat zprávu ESB službě, jejíž instance jsou vystaveny na více fyzických uzlech, vzniká otázka, která konkrétní instance služby se vybere. JBossESB nabízí několik vestavěných triviálních vyvažovacích politik:

- **RandomRobin**: Tato politika ze seznamu adres dané služby vrátí náhodnou adresu.
- **RoundRobin**: Politika cyklicky projíždí seznam adres všech instancí dané služby. Vrátí adresu služby, v dalším volání vrátí další adresu atd.
- **FirstAvailable**: Politika ze seznamu adres vezme náhodnou adresu. Politika pak opětovně používá tuto adresu do té doby, dokud je tato adresa platná. V této politice tedy vypínáme vyvažování zátěže.

Kromě těchto politik je možné doprogramovat a integrovat do JBossESB libovolnou jinou politiku.

Základním předpokladem pro použití vyvažování zátěže je to, že služby musí být **bezestavové**. Bezestavovost služby znamená, že ESB služba si neuchovává žádný kontext mezi dvěma zpracovávanými zprávami. Příčinou je vyvažování zátěže, které může dvě po sobě následující zprávy poslat dvěma rozdílným instancím stejné služby.

7.2.3 Transakční posílání zpráv

Pokud je jako transportní protokol používáno JMS, tak je možné poslání JMS

zprávy zahrnout do aktuální transakce. Zpráva se odešle jen tehdy, pokud se povede potvrdit transakci. Tento transakční přístup poskytovatele je velice důležitý v momentě, kdy je zpráva posílána v nějakém větším kontextu prací, které se musí provést všechny, nebo žádná. Příkladem může být poslání několika zpráv, kdy je potřeba, aby byly poslány všechny zprávy, nebo žádná z nich.

7.2.4 Transakční zpracování zpráv

Může se stát, že se provedení business logiky v ESB službě nepovede z důvodu nějaké dočasné chyby (interně identifikováno potomkem výjimky `java.lang.RuntimeException`). Touto chybou může být např. výpadek spojení s databází. Při dalším pokusu se již zpracování může podařit. Pokud dojde k nějaké dočasné chybě, je zpráva opětovně předána ke zpracování.

ESB služba může být nakonfigurována následujícími parametry:

- Maximálním počtem pokusů o zpracování zprávy.
- Časovým limitem, do jehož vypršení je nutné zprávu zpracovat. Jinak se JBossESB pokusí zprávu zpracovat znova.
- Zpožděním mezi dvěma pokusy o zpracování zprávy.

Pokud se vyčerpá počet pokusů na zpracování zprávy, tak je poslána standardní chybová zpráva službě, jejíž jméno je nalezeno v metadatech vstupní zprávy.

7.2.5 Zapojení business logiky do ESB služby

Business logika je zapojena do akce v action processing pipeline. Akce je Java třída.

Z akce je možné zavolat:

- session beanu bežící na stejném/jiném aplikačním serveru,
- běžnou Java třídu,
- jakoukoli heterogenní službu pomocí Java knihovny,
- ...

Business logika systému Club je napsaná ve stateless session beanách [6]. Akce služby si pomocí JNDI [37] vyžádá **lokální** referenci na danou session beanu. Tu potom může zavolat. Autor diplomové práce využívá toho, že ESB služba běží ve stejném adresovém prostoru (ve stejném aplikačním serveru) jako session beana. Proto stačí zavolat session beanu prostřednictvím reference a není potřeba další serializace a deserializace argumentů. Každá serializace a deserializace je časově velmi drahá operace.

Nějaká osoba musí naimplementovat akci volající business logiku. Ačkoli to není nic složitého, základní znalosti o podnikové sběrnici JBossESB mít musí. Pro účely diplomové práce je tato osoba nazvána *Programátor ESB*. V akci se pouze volá business

logika a nenachází se tam žádná komunikační logika (odeslání odpovědi apod.). Programátor ESB musí znát:

- API business logiky,
- odpovědnost služby,
- programovací jazyk Java,
- způsob, kterým se z akcí skládají ESB služby (viz JBossESB Programmers Guide [38], kapitola Building and Using Services).

7.2.6 Management

JBossESB poskytuje administrátorské rozhraní, kterým lze u každé instance služby sledovat:

- počet úspěšně, neúspěšně zpracovaných zpráv,
- délku trvání jednotlivých akcí v action processing pipeline,
- celkovou délku trvání zpracování zprávy.

Naměřené hodnoty jsou přehledně v čase zobrazovány do grafu.

Z výpisů je možné zjistit výkonnost jednotlivých uzlů podnikové sběrnice. Dále je možné zjistit služby, které nejdéle zpracovávají vstupní zprávu. Monitorování provozu je důležité pro nacházení úzkých míst v návrhu BP.

7.3 Integrace business procesu do podnikové sběrnice služeb JBossESB

Integrace BP do podnikové sběrnice služeb se skládá z následujících kroků:

1. **Namodelování business procesu:** Architekt BP namodeluje BP v nástroji jBPM GPD. Architekt by měl algoritmus dekomponovat na části tak, aby části byly:
   - vhodně granulované,
   - bezestavové (stateless),
   - pokud možno nezávislé a současně vykonavatelné.
2. **Implementace jednotlivých částí algoritmu:** Aplikační programátor naimplementuje (např. v EJB 3.0) business logiku jednotlivých částí algoritmu tak, jak je navrhl architekt.
3. **Integrace business logiky do služeb**: Programátor ESB zapojí business logiku do služeb. Z každé části algoritmu tak vznikne služba.
4. **Nasazení služeb do podnikové sběrnice**: Programátor ESB nasadí služby na jednotlivé uzly podnikové sběrnice. Programátor ESB zvolí vhodný počet nasazovaných instancí služeb. Přesný počet instancí nasazených na jeden počítač

se liší podle typu služby, výkonnosti počítače, počtu procesorů a procesorových jader daného počítače.

5. **Nahrání namodelovaného BP do procesního manažeru**: Architekt BP nahraje namodelovaný BP do procesního manažeru.

7.4 Shrnutí výhod a nevýhod JBossESB

JBossESB je distribuovaný škálovatelný framework, který společně s procesním manažerem jBPM poskytuje skvělé podmínky pro distribuované zpracování business procesů.

Výhody použití JBossESB:

- Jako transportní protokol lze využívat JMS, ale i jiné transportní protokoly (např. komunikaci přes soubory, databázi).
- JBossESB má integrovaný registr služeb.
- Je velmi snadné rozšířit podnikovou sběrnici služeb o další fyzický uzel.
- JBossESB je integrovaná s procesním manažerem jBPM [29]. Procesní manažer se stará o koordinaci služeb. Procesní manažer umožňuje business proces řídit z administrátorského rozhraní. Rozhraní administrátora BP umožňuje:
  - spustit BP,
  - pozastavit BP,
  - opětovně rozběhnout BP,
  - předčasně ukončit BP.
- JBossESB podporuje transakční zpracování zpráv:
  - Odeslání zprávy je součástí aktuální transakce.
  - Pokud se zpracování zprávy nezdaří a příčinou je dočasná chyba, je zpráva znovu předána ke zpracování.
- JBossESB odděluje business logiku od komunikační logiky. O komunikaci se podniková sběrnice služeb stará sama. Aplikační programátor implementuje pouze business logiku služeb a o propojení služeb se nestará.
- JBossESB umožňuje do business procesu transparentně začlenit heterogenní službu (např. webovou službu).
- JBossESB je součástí aplikačního serveru JBoss. Business logika, která je nasazena v aplikačním serveru, může být z ESB služby vyvolána přímo referencí, není potřeba další serializace a deserializace.

Nevýhody použití JBossESB:

- Nasazením JBossESB musí ve firmě existovat osoba, která podnikové sběrnici rozumí – programátor ESB. Programátor ESB má za úkol zejména integraci business logiky do ESB služeb.
- Implementace JBossESB se neustále vyvíjí. Tento vývoj s sebou nese množství zanesených implementačních chyb. Autor s podnikovou sběrnicí pracuje přes rok. Poslední releasy již tolik chyb neobsahují a jsou stabilní.

8 Volba a integrace distribuovaných transakcí do JBossESB

Paralelizací algoritmu získáme části algoritmu, které mohou běžet současně. Většinou je nutné, aby tyto části pracovaly v rámci jedné transakce. Vzhledem k tomu, že tyto části mohou běžet na různých počítačích, je nutné, aby transakce byla distribuovaná.

Distribuované (globální) transakce lze podle jejich délky trvání rozdělit na:

- **Krátkodobé**: Krátkodobé distribuované transakce jsou v podstatě klasické flat transakce běžící v distribuovaném prostředí [39]. Tyto transakce splňují ACID [40] vlastnosti. Těchto základních vlastností je často dosahováno tak, že transakce drží na datech databázové zámky po celou dobu trvání transakce. Jednotlivé transakční zdroje jsou koordinovány transakčním manažerem pomocí dvoufázového commit protokolu [41] (2PC). J2EE poskytuje možnost využívání krátkodobých distribuovaných transakcí prostřednictvím Java Transaction Service [18] (JTS).

- **Dlouhodobé**: Pokud transakce trvá hodiny až dny, tak není vhodné držet databázové zámky na datech po celou dobu trvání distribuované transakce. Dlouhodobá transakce může být vnímána jako sekvence databázových transakcí, které jsou seskupené do jedné atomické jednotky [42]. Obvykle tyto transakce využívají princip kompenzačních transakcí [43] pro odvolání změn.

JBossESB nenabízí přímou podporu distribuovaných transakcí. Autor diplomové práce musel zvolit jistou implementaci distribuovaných transakcí a integrovat ji do podnikové sběrnice služeb. V prostředí J2EE aplikačního serveru JBoss existují dvě cesty, jak docílit distribuovaných transakcí [49]:

- použít distribuovaný transakční systém s podporou JTS,
- použít distribuované transakce koordinované rodinou protokolů WS-Transaction [44].

8.1 Distribuovaný transakční systém v prostředí J2EE

Distribuovaný transakční systém se v prostředí J2EE skládá z několika částí [45]:

- **Aplikační server** poskytuje aplikacím transakční běhové prostředí.

- **Resource manažer** skrz resource adaptér poskytuje aplikaci přístup k XA zdrojům. Příkladem resource manažeru je relační databáze a příkladem resource adaptéru je JDBC driver. XA zdroj je zdroj podporující dvoufázový commit protokol.

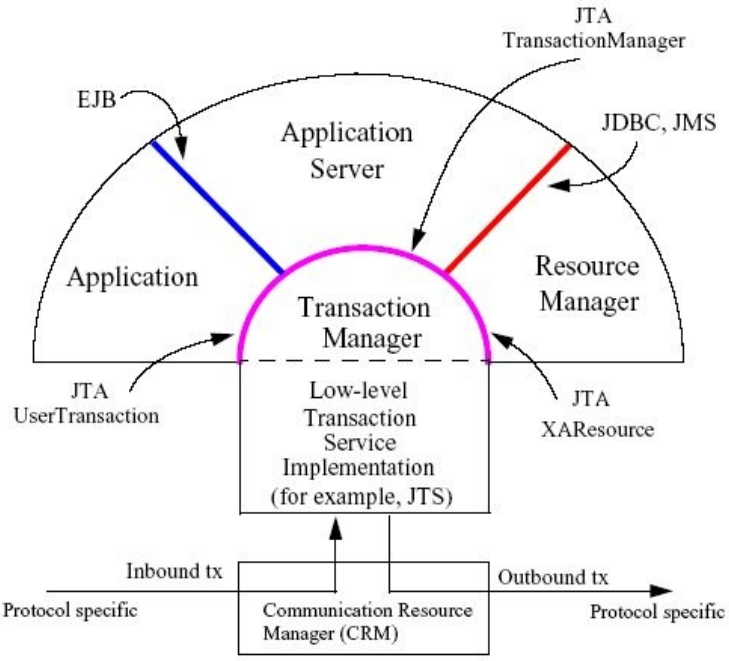
- **Aplikace** vyžadující transakční služby aplikačního serveru.

- **Transakční manažer** poskytuje služby a operace pro zahájení a ukončení transakce, management XA zdrojů apod. Je využíván aplikačním serverem.

- **Communication resource** manažer umožňuje propagaci transakčního kontextu

mezi více transakčními manažery a obsluhuje příchozí a odchozí požadavky na transakční zpracování.

Obrázek 8 zobrazuje, jak spolu souvisí jednotlivé části distribuovaného systému. Menší z polokružnic je Java Transaction API [45] (JTA). JTA definuje lokální Java rozhraní mezi transakčním manažerem a částmi v distribuovaném transakčním systému: aplikací, manažerem zdrojů a aplikačním serverem.



Obrázek 8: Entity v distribuovaném transakčním systému

8.1.1 Java Transaction Service

Java Transaction Service [18] (JTS) je možná implementace JTA transakčního manažeru. JTS mapuje standard CORBA [46] pro transakce OTS 1.1 [47] do Javy. OTS 1.1 umožňuje prostřednictvím protokolu IIOP [48] propagaci transakcí mezi dalšími transakčními manažery podporujícími OTS 1.1. JTS však nestanovuje pravidla pro komunikaci na bázi protokolu IIOP. To zajišťuje tzv. communication resource manager, jehož hlavní funkcí je právě propagace transakčního kontextu mezi transakčními manažery a obsluha příchozích a odchozích požadavků na transakční zpracování.

8.1.2 Setkání s realitou v prostředí aplikačního serveru JBoss

JTS je mocný nástroj a je možným řešením, jak integrovat distribuované transakce do podnikové sběrnice služeb. Samozřejmě za předpokladu, že by jednotlivé služby využívaly služeb OTS 1.1 transakčních manažerů.

Záměrem autora bylo využití služeb JTS v ESB. Integraci JTS do ESB ale zabránila informace ze stránek projektu JBoss Transactions [76] týkající se balíku

knihoven JTS [49]:

> *It's not suitable for use with JBoss 4.2.x or the EAP (JBoss Enterprise Application Platform – pozn. autora) due to changes in the version of JacORB they use. If you require JTS for the EAP please contact JBoss support.*

JBoss support je placená podpora. Autor diplomové práce nemůže využít placené podpory ani služeb aplikačního serveru JBoss verze 5.0 podporujícího JTS, protože na něm nelze spustit JBossESB 4.3.GA (testováno autorem). Dokumentace k JBossESB 4.3.GA uvádí [50], že doporučený aplikační server, na který lze JBossESB nasadit, je JBossAS 4.2.2.

Autor diplomové práce došel k závěru, že schůdným řešením je integrace distribuovaných transakcí koordinovaných rodinou protokolů WS-Transaction. Jeden z hlavních důvodů, proč autor použil WS-Transaction, bylo, že v době vytváření frameworku to byla jediná funkční implementace distribuovaných transakcí, která šla nasadit na aplikační server JBoss 4.2.2.

8.2 Rodina standardů WS-Transaction

Ian Robinson uvádí [51], že rodina standardů Web Service Transaction definuje mechanismy, které umožňují začlenit práci webové služby do distribuované transakce.

Sada standardů WS-Transaction byla vytvořena společnostmi IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA. Do této rodiny se řadí standardy:

- WS-Coordination (WS-C),
- WS-AtomicTransaction (WS-AT),
- WS-BusinessActivity (WS-BA).

8.2.1 WS-Coordination (WS-C)

WS-Coordination [52] definuje rozšiřitelný framework pro koordinovaní aktivit za použití koordinátoru a množiny dalších protokolů. Distribuovaní účastníci transakce jsou schopni použít WS-Coordination k seskupení svých akcí dohromady, a dosáhnout tak jednotného výsledku.

WS-Coordination definuje kontext, který může být považován za identifikátor, pod kterým jsou aktivity vykonávány.

WS-Coordination je ale pouze framework. Způsob, jakým se dosáhne jednotného výsledku účastníků, definují standardy WS-AtomicTransaction a WS-BusinessActivity.

8.2.2 WS-AtomicTransaction (WS-AT)

Specifikace WS-AtomicTransaction [53] říká, že účastníci globální WS-AT transakce se koordinují pomocí dvoufázového commit protokolu (2PC). Tato specifikace definuje dva hlavní protokoly, které říkají, jakým způsobem se dosáhne

konsensu mezi účastníky globální transakce:

- durable 2PC (klasický dvoufázový commit protokol),
- volatile 2PC.

Volatile 2PC protokol se spíše hodí na zdroje, které podléhají časté změně, např. na data uložená ve vyrovnávací paměti. Naproti tomu durable 2PC protokol se hodí v případě, že je třeba koordinovat persistentní zdroje jako např. databáze. Koordinátor provádí oba protokoly zároveň s tím omezením, že *prepare* fáze volatile 2PC protokolu předchází prepare fázi *durable* 2PC protokolu. Není garantováno, že volatile účastník obdrží notifikaci o výsledku globální transakce. Dobrým příkladem volatile účastníka je vyrovnávací paměť, která se v momentě prepare fáze sesynchronizuje s databází.

Každý účastník globální transakce pracuje v kontextu své vlastní atomické transakce. Atomickými transakcemi jsou míněny klasické transakce s ACID vlastnostmi. Těchto základních vlastností je nejčastěji dosahováno použitím databázových zámků. Je vhodné, aby atomické transakce byly kvůli zmíněnému držení zámků co nejkratší. Tuto atomickou transakci si účastník vytvoří v momentě, kdy se registruje ke globální transakci. Atomická transakce tedy žije po dobu života účastníka. Čím delší je globální transakce, tím delší jsou jednotlivé atomické transakce držící databázové zámky.

8.2.3 WS-BusinessActivity (WS-BA)

WS-BusinessActivity [54] je rozšířeným transakčním modelem vytvořeným za účelem podpory dlouhotrvajících business procesů. U tohoto standardu se místo označení globální transakce užívá spíše termín business aktivita. Business aktivita nepoužívá pro zaručení konsensu mezi účastníky dvoufázový commit protokol, ale používá kompenzační model pro odvolání změn. To protokol WS-BA činí více vhodným pro dlouhotrvající procesy. Avšak vyžaduje to uvolnění některých ACID vlastností transakcí, zejména izolaci.

Programování business aktivit většinou zahrnuje více úsilí než je potřeba v případě použití WS-AtomicTransaction. Jde hlavně o kompenzační logiku, která musí být manuálně přidána do business logiky pro umožnění odvolání změn.

U standardu WS-AtomicTransaction byly atomické transakce aktivní od doby zaregistrování účastníka ke globální transakci, až do doby, kdy byla globální transakce ukončena. U standardu WS-BusinessActivity může být život atomických transakcí o mnoho kratší. Jakmile účastník dokončí svou práci, může atomickou transakci potvrdit. Pro případ budoucího odvolání změn má k dispozici kompenzační transakci. Účastník vždy informuje koordinátora o tom, jak dokončil svou práci (úspěšně, nebo neúspěšně). V momentě, kdy nějaká entita požádá o skončení business aktivity, koordinátor zajistí následující:

- Pokud všichni účastníci dokončili svou práci úspěšně, je business aktivita skončena.
- Pokud některý z účastníků skončil neúspěšně, požádá o provedení kompenzačních transakcí všechny účastníky business aktivity.

8.2.4 Konkurenční přístup k datům – uzamykací modely

Tato podkapitola popisuje obecné metody konkurenčního přístupu k datům, které se nijak netýkají rodiny protokolů WS-Transaction.

Řízení přístupu k datům zajišťuje, že databázové transakce mohou být vykonávány na stejných datech současně bez porušení integrity databáze.

Existují dva hlavní modely přístupu k datům [55]:

- **Pesimistický**: V pesimistickém modelu transakce zamyká data před tím, než k nim přistoupí, a neuvolní je, dokud transakce neskončí.
- **Optimistický**: V optimistickém modelu jsou zámky získány těsně před čtecí operací a uvolněny okamžitě po ní. Update zámky jsou získané těsně před update operací a drženy, dokud transakce neskončí.

Optimistické zamykání podporuje vysokou paralelitu a škálovatelnost. Výhody optimistického zamykání jsou následující [56]:

- Záznamy jsou zamčeny po kratší dobu než v pesimistickém modelu.
- Data, která jsou přečtena pro update, jsou zamčena až od té doby, co je update skutečně proveden, a jsou uvolněna až při ukončení transakce. Pesimistický přístup zamyká data okamžitě, když k nim přistoupí.

Nevýhodou optimistického zamykání je, že potvrzení transakce A se nemusí podařit z toho důvodu, že nějaká jiná transakce B pozměnila stejný záznam od doby, co ho transakce A načetla. Této situaci se říká kolize. Transakce A musí být opakována, což může vést ke snížení výkonu [56].

Pesimistické metody se používají tehdy, když předpokládáme, že bude docházet ke konfliktům při přístupu jednotlivých transakcí ke shodným datům. Optimistické metody používáme v opačném případě.

8.2.5 Vlastnosti transakcí koordinovaných rodinou protokolů WS-Transaction

Výhody transakcí postavených na standardech rodiny WS-Transaction:

1. Jeden transakční systém může spolupracovat s jiným transakčním systémem.
2. Příležitost, jak využít možnosti skryté paralelity business procesů skládajících se ze služeb heterogenních systémů.
3. WS-BA umožňuje vypořádání se s dlouhotrvajícími aktivitami přístupem podobným v Open nested transactions [57] za využití kompenzačních transakcí, což zvyšuje paralelitu systému (zámky na databázových zdrojích jsou drženy krátce).

Tabulka 1 srovnává standardy WS-BA a WS-AT.

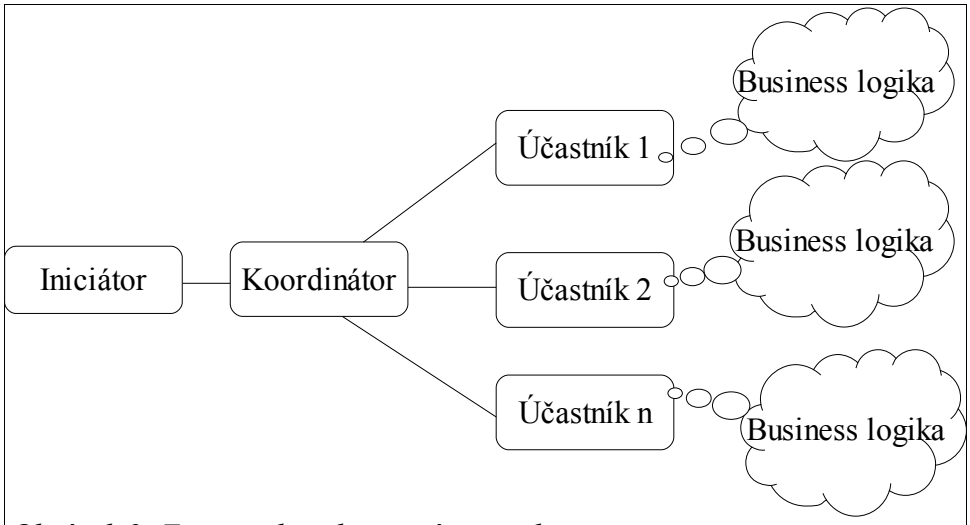
WS-BusinessActivity	**WS-AtomicTransaction**
Vhodné pro dlouhotrvající (hodiny, dny) procesy.	Vhodné pro procesy trvající krátce.
Práce business procesu je rozložena do mnoha malých atomických transakcí, které trvají krátce. Okolní svět vidí mezivýsledky business procesu (je narušena vlastnost izolace transakcí).	Celý business proces běží v jedné distribuované transakci. Žádné mezivýsledky nejsou propagovány vnějšímu světu (vlastnost izolace transakcí je naplňována).
Je potřeba programovat kompenzační logiku. Kompenzační logika může být velmi složitá. Vzniká problém, pokud se kompenzační transakce nepovede provést.	Není potřeba programovat kompenzační logiku.
Nedrží zámky na databázových zdrojích po dlouhou dobu.	Drží zámky na databázových zdrojích po celou dobu trvání globální transakce. Použitím optimistického zamykání můžeme tuto dobu zkrátit a omezit pouze na update zámky.

Tabulka 1: Srovnání WS-BA a WS-AT

8.3 Integrace standardů WS-Transaction do JBossESB

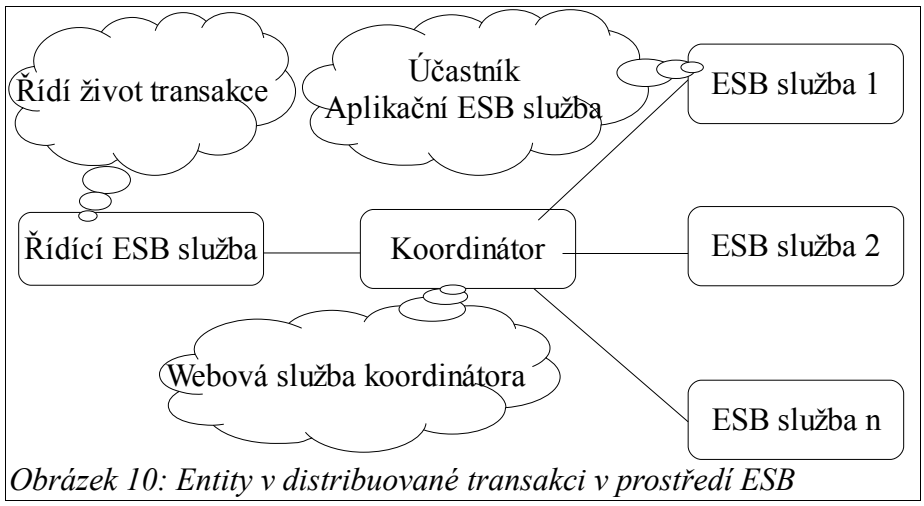
Ať už používáme koordinační typ WS-AT nebo WS-BA, definujeme v transakci resp. business aktivitě několik rolí (viz obrázek 9) – autor užívá pro globální transakci resp. business aktivitu společný termín distribuovaná transakce:

- **Iniciátor**: Iniciuje požadavky na začátek a ukončení distribuované transakce.
- **Koordinátor**: Koordinuje distribuovanou transakci podle použitého koordinačního protokolu.
- **Účastník**: Registrací k distribuované transakci vzniká její účastník. Účastník je zodpovědný za vykonání koordinačních požadavků (prepare, commit apod.). Podle použitého koordinačního protokolu musí koordinátorovi po vykonání požadavku odpovědět.



Obrázek 9: Entity v distribuované transakci

Cílem autora práce bylo vytvořit ESB služby plnící roli iniciátora, které by šlo začlenit do návrhu business procesu v jBPM GPD. Dalším cílem bylo umožnit ESB službě pracovat v rámci globální transakce resp. business aktivity (viz obrázek 10). Každá zpráva, jejíž obsah má být zpracován v rámci nějaké distribuované transakce, musí obsahovat kontext (identifikátor) této distribuované transakce. Každá ESB služba zná adresu koordinátora, aby skrze něj zaregistrovala účastníka ke konkrétní distribuované transakci.



Obrázek 10: Entity v distribuované transakci v prostředí ESB

Řídící služba (iniciátor) a služba účastníka využívají klientskou knihovnu projektu JBoss Transactions, pomocí níž delegují svá volání na koordinátora. Koordinátor je webová služba. Umožňuje koordinovat účastníky pomocí WS-AT i WS-BA. Implementaci koordinátora autor převzal z projektu JBoss Transactions.

Autor integroval do JBossESB možnost využití standardu WS-AtomicTransaction i WS-BusinessActivity. Řídící služby jsou rozděleny do dvou skupin. První skupina řídí globální transakce koordinované pomocí standardu WS-AtomicTransaction a druhá skupina řídí business aktivity koordinované standardem WS-BusinessActivity. Všechny

služby znají síťovou adresu koordinátora. Každá zpráva, která má být zpracována v rámci jisté globální transakce či business aktivity, musí obsahovat její kontext (identifikátor).

8.3.1 ESB Služby pracující se standardem WS-AtomicTransaction

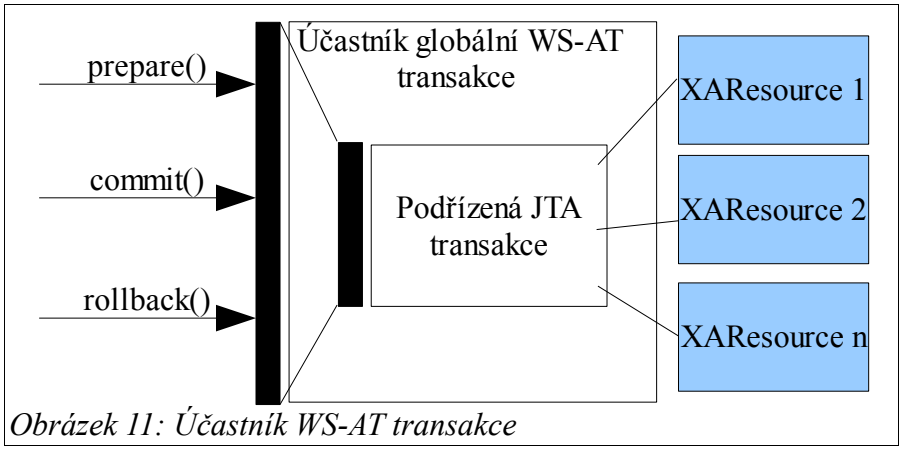
V případě standardu WS-AT je odpovědnost řídící služby rozdělena mezi následující ESB služby:

- **WSATxBeginTransaction**: Vyvoláním této služby vznikne globální transakce. Kontext (identifikátor) vzniklé transakce je vložen do odpovědi.
- **WSATxTerminateService**: Vyvoláním této služby se potvrdí globální transakce identifikovaná kontextem v příchozí zprávě.
- **WSATxRollbackService**: Vyvoláním této služby se odvolají změny způsobené globální transakcí identifikovanou kontextem v příchozí zprávě.

Služby znají síťovou adresu koordinátora a komunikují s ním pomocí protokolu SOAP.

8.3.2 Zpracování ESB zprávy účastníkem globální transakce

Účastník globální transakce musí být nějak namapován na transakční zdroj – XA resource. Účastník deleguje zprávy dvoufázového commit protokolu (2PC) *commit*, *prepar*e a *rollback* na tento transakční zdroj. Zprávy 2PC ovšem nejsou delegovány na transakční zdroje přímo. Zprávy 2PC protokolu jsou nejdříve mapovány na atomickou podřízenou (nested) JTA transakci (viz obrázek 11). Podřízená je proto, že na ní lze volat operace *commit*, *prepare*, a *rollback*. Tato podřízená transakce ovládá několik transakčních zdrojů. V rámci této podřízené transakce jsou prováděny všechny databázové operace v business logice ESB služby, které se mají provést v rámci globální transakce.



Obrázek 11: Účastník WS-AT transakce

Aby byla činnost ESB služby asociována s globální transakcí, musí ESB služba při přijetí zprávy provést následující kroky:

1. Přečíst kontext globální transakce ze vstupní zprávy.

2. Pokud neexistuje mapování na podřízenou transakci, vytvořit podřízenou transakci, zaregistrovat nového účastníka globální transakce u koordinátora, zaznamenat mapování <globální transakce, podřízená transakce>; jinak na základě mapování získat odpovídající podřízenou transakci. Toto mapování umožňuje přemostění mezi globální transakcí a podřízenou transakcí.
3. Vykonat business logiku v rámci podřízené transakce.

8.3.3 Služby pracující se standardem WS-BusinessActivity

V případě standardu WS-BA je odpovědnost řídící služby rozdělena mezi následující ESB služby:

- **WSBABeginService**: Vyvoláním této služby vznikne business aktivita. Kontext (identifikátor) vzniklé business aktivity vloží do odpovědi.
- **WSBACloseService**: Vyvoláním této služby se ukončí (potvrdí) business aktivita identifikovaná kontextem v příchozí zprávě.
- **WSBACancelService**: Vyvoláním této služby se odvolají změny způsobené business aktivitou identifikovanou kontextem v příchozí zprávě.

Služby znají síťovou adresu koordinátora a komunikují s ním pomocí protokolu SOAP.

8.3.4 Zpracování ESB zprávy účastníkem business aktivity

ESB služba, která chce svou činnost vykonávat v rámci business aktivity, musí při přijetí zprávy provést následující kroky:

1. Přečíst kontext business aktivity ze vstupní zprávy.
2. Vytvořit novou atomickou transakci.
3. Vykonat business logiku v rámci vytvořené atomické transakce.
4. Vytvořit účastníka business aktivity. Účastník je nainicializován kompenzační logikou, která bude zavolána v případě, že bude nutné odvolat změny, které atomická transakce způsobila.
5. Zaregistrovat účastníka k business aktivitě identifikovanou přečteným kontextem. ESB služba registruje účastníka business aktivity u koordinátora.
6. Potvrdit atomickou transakci.
7. Informovat koordinátora o výsledku (úspěch/neúspěch) provedení business logiky.

8.4 Zotavení po výpadku koordinátora nebo účastníka

Současná implementace rodiny standardů WS-Transaction neobsahuje modul

zotavení po výpadku koordinátora nebo účastníka distribuované transakce. Autor diplomové práce převzal implementaci WS-Transaction z projektu JBoss Transactions.

8.4.1 Zotavení po výpadku WS-Transaction koordinátora

Současná implementace neumožňuje zotavení po výpadku koordinátora. Koordinátor uchovává všechny informace o účastnících globální transakce respektive business aktivity v paměti. Tyto informace jsou po jeho výpadku nenávratně ztraceny.

V zásadě existuje několik možností, jak docílit zotavení po výpadku koordinátora:

- Použít JTS v momentě, kdy JBossESB půjde nasadit na aplikační server JBoss verze 5.0. V JTS je zotavení po výpadku koordinátora funkční.
- Počkat na vydání implementace [58] sady standardů WS-Transaction, které bude obsahovat modul zotavení po výpadku koordinátora.
- Dopsat do implementace koordinátora tvorbu žurnálu tzn. především seznam účastníků a stavy, ve kterých se nacházejí. Po opětovném nastartování koordinátor žurnál přečte a dostane se tak do stavu, ve kterém byl před svým výpadkem.

8.4.2 Zotavení po výpadku účastníka WS-BA a WS-AT

V případě WS-BA je účastník nainicializován informacemi, které jsou nutné pro vykonání kompenzační transakce. V případě výpadku účastníka dojde k nenávratné ztrátě těchto informací.

Pokud spadne účastník WS-AT transakce, tak koordinátor globální transakci odvolá. Problém ale nastane, když účastník spadne poté, co se dostane do stavu *prepared* a předtím, než dostane příkaz *commit*. Tento účastník zůstane ve stavu *prepared* v databázi jako zombie do té doby, dokud ho manuálně administrátor databáze neodvolá.

V případě havárie účastníka WS-AT je možné následující řešení. Pokud účastník zhavaruje během *prepared* stavu, tak se po svém opětovném startu domluví s koordinátorem, zda-li má svou práci potvrdit, nebo odvolat.

Pokud spadne účastník WS-BA, tak je nutné obnovit data, kterými byla nainicializována kompenzační logika. Po svém nastartování se účastník domluví s koordinátorem, zda-li má zavolat kompenzační transakci.

8.4.3 Výpadek sítě

Autor testoval implementaci standardů WS-AT a WS-BA a simuloval nedostupnost koordinátora a jednotlivých účastníků. Protokoly WS-BA a WS-AT jsou naimplementovány tak, aby přestály výpadek sítě.

Vzhledem k tomu, že infrastruktura je navržena do prostředí lokální sítě, výpadky by měly být velmi vzácné.

9 Analýza zpracování BP ve vytvořené infrastruktuře

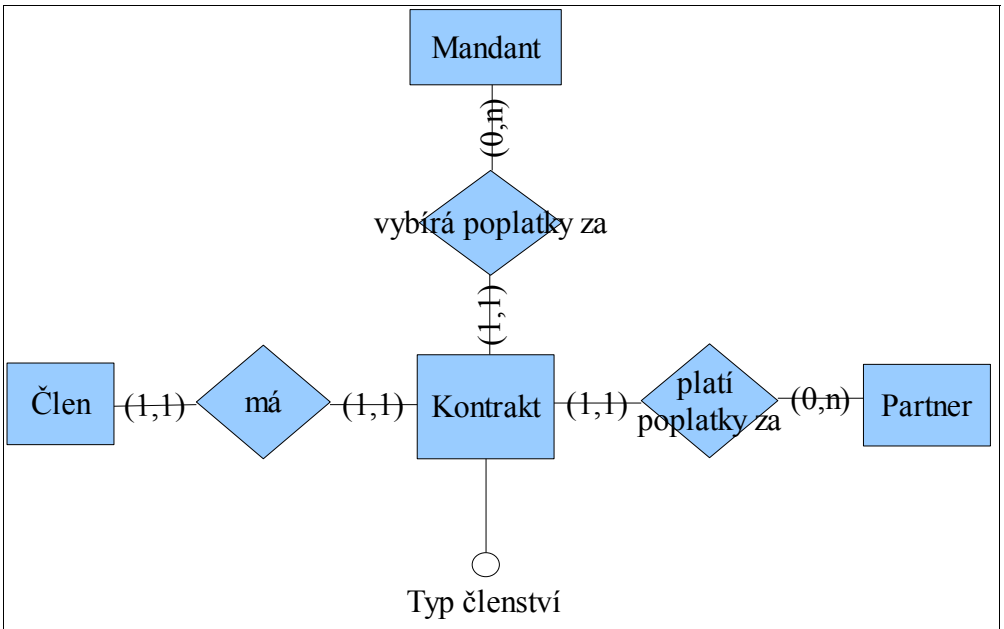
V této kapitole je popsáno nasazení konkrétního business procesu *Member fees* na infrastrukturu tvořenou podnikovou sběrnicí služeb. Proces *Member fees* je součástí posloupnosti business procesů, které administrátor BP spouští ve správném pořadí na konci každého měsíce.

9.1 Kontext business procesu Member fees

Jak již bylo uvedeno v kapitole 3.1, ve věrnostním systému Club se nachází několik rolí:

- **Člen klubu**: Osoba, která vlastní klubovou kartu.
- **Partner klubu**: Podnik přímo poskytující služby členům klubu jako např. čerpací stanice, autolakovna, autoopravna apod.
- **Mandant klubu**: Výrobce automobilů, náhradních dílů a doplňků. Vydavatel klubového magazínu, ve kterém partneři klubu nabízejí své služby. V současnosti existují dva mandanti[2]: Mandant1 a Mandant2.

Za členství v klubu vybírá mandant poplatky. Poplatek za členství si jedinec může hradit sám, ale většinou za něj členský poplatek hradí některý z partnerů mandanta. Tomuto vztahu mezi členem, partnerem a mandantem se říká kontrakt. Partner tím rozšiřuje **svou** členskou základnu (zakládá nové kontrakty), protože čím více členů má, tím větší prostor pro reklamu dostává v magazínech, které rozesílají mandanti členům klubu. Vztah mezi členem, partnerem a mandantem je zobrazen na obrázku 12.



Obrázek 12: ER diagram entit v systému

Proces *Member fees* vypočítávává poplatky partnerů za členství zákazníků v

2 Licenční podmínky a zabezpečení neumožňují zveřejnění jmen automobilových značek

klubu. Partner hradí poplatek za členství jen těch zákazníků, se kterými má kontrakt. Výše poplatku se liší podle typu členství. Existuje:

- běžné členství,
- členství pro zaměstnance partnera klubu,
- členství pro VIP zákazníky,
- členství pro pracovníky autoškol,
- …

Algoritmus business procesu *Member fees*:

Přečti z databáze seznam všech typů klubových členství.

for each typ členství **do**

Přečti z databáze všechny validní kontrakty daného typu členství.

for each kontrakt **do**

Vybraný kontrakt zpoplatni. Výši poplatku vypočti na základě typu členství. Plátcem členství je vlastník kontraktu, tj. partner. Záznam o zpoplatnění vlož do databáze.

Velikost vstupních dat:

- počet klubových členství ~ 30
- počet validních kontraktů se liší podle zpracovávaného typu členství
  - běžný typ členství pro Mandant1 ~ 1 200 000 kontraktů
  - běžný typ členství pro Mandant2 ~ 500 000 kontraktů
  - autoškoly ~ 2 200 kontraktů
  - VIP členství ~ 1 300 kontraktů
  - …

Velikost výstupních dat:

- ~ 800 000 záznamů o zpoplatnění kontraktů

Celkový počet kontraktů, které je nutné přečíst z databáze, je přibližně 2 000 000. Z každého kontraktu vzniká maximálně jeden záznam o zpoplatnění. Záznamu o zpoplatnění kontraktu se říká Club transakce. Některé kontrakty zpoplatněny nejsou, ale je **nutné** je z historických důvodů přečíst. Celkově je potřeba zapsat do databáze přibližně 800 000 Club transakcí.

9.2 Integrace Member fees do distribuované infrastruktury

Kapitola 7.3 uvádí postup, jakým obecně překlápět business procesy do podnikové sběrnice služeb.

9.2.1 Diskuse návrhu definice procesu *Member fees*

Autor práce diskutuje tři návrhy definice *Member fees*. Postupuje od naivního návrhu k návrhu, který je použitelný.

Definice procesu na obrázcích 13, 14 a 15 obsahují dva typy uzlů:

- **Uzly řídící**: Vytvářejí základní strukturu definice business procesu. Jsou to např. počáteční, koncový uzel, for cyklus, splitter [59], aggregator [60]. Chyba v těchto typech uzlů je považována za fatální a neočekává se.
- **Uzly obsahující business logiku**: To jsou uzly reprezentující většinou volání ESB služby. Pokud se v ESB službě vyskytne chyba, tak ESB služba pošle chybovou zprávu zpět procesnímu manažeru a procesní manažer se pokusí odvolat všechny změny v databázi, které business logika procesu provedla.

Autor práce dekomponoval business logiku procesu *Member fees* na tři služby:

- Služba *ReadScheduledContractTypesService*: Přečte všechny typy členství. Služba je během procesu *Member fees* zavolána pouze jednou.
- Služba *ContractsTypesService*: Přečte všechny identifikátory validních kontraktů daného typu členství. Služba je pro každý typ členství zavolána jednou. Musí přečíst identifikátory záznamů kontraktů, protože všechny záznamy kontraktů se nevejdou do paměti.
- Služba *ContractsProcessorService*: **Sekvenčně** zpracuje kolekci identifikátorů kontraktů. Tato služba je během procesu *Member fees* zavolána mnohokrát. Na každém uzlu podnikové sběrnice se nachází několik instancí této služby. Tyto instance jsou vzájemně **nezávislé** a mohou vykonávat přidělenou práci **současně**.
  Velikost kolekce identifikátorů, která je službě předána, určuje granularitu této služby. Čím větší kolekce je, tím více kontraktů služba zpracuje během svého jednoho vyvolání.

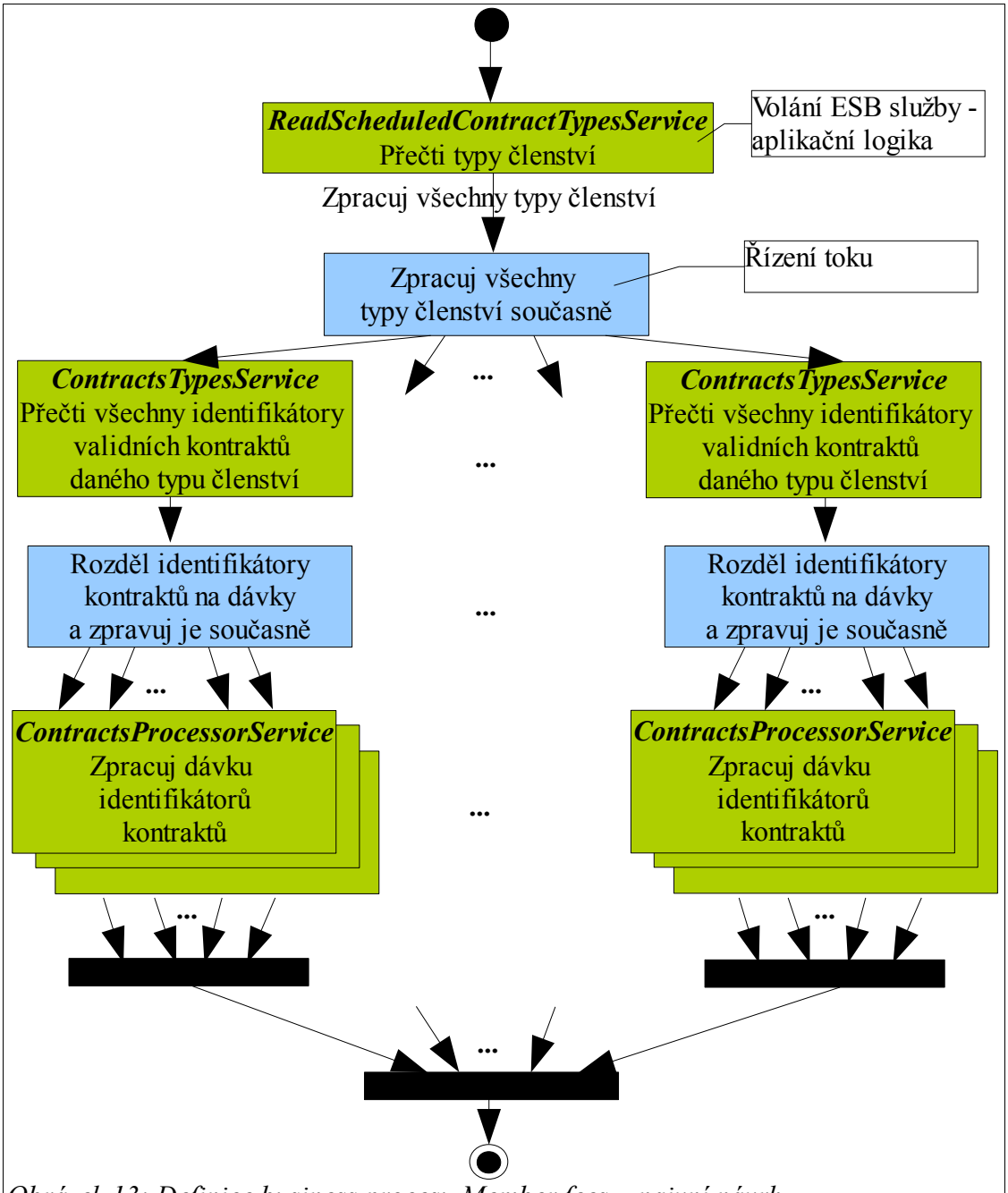
Algoritmus procesu *Member fees* je vysoce paralelizovatelný. Můžeme volit různou míru paralelizace:

- Můžeme všechny typy členství zpracovávat současně.
- Můžeme načtené identifikátory kontraktů rozdělit na dávky a všechny dávky zpracovávat současně.
- Můžeme načtené identifikátory kontraktů rozdělit na dávky a dávky zpracovávat jednu po druhé. Samotnou dávku je možné rozdělit na části a tyto části zpracovávat současně.

9.2.1.1 Definice business procesu Member fees – naivní návrh

První návrh procesu *Member fees* zakreslený na obrázku 13 se snaží maximálně využít paralelity algoritmu:

- Jednotlivé typy členství se zpracovávají současně.
- Identifikátory daného typu členství se rozsekají na dávky a tyto dávky se zpracovávají současně.



Obrázek 13: Definice business procesu Member fees – naivní návrh

Výhody návrhu:

- Výhoda návrhu je především jednoduchost a srozumitelnost.

Nevýhody návrhu:

- Business proces není možné pozastavit. V momentě, kdy procesní manažer pošle požadavky na zpracování dávek, tak již business proces nelze pozastavit ani přerušit, protože požadavky se již zpracovávají (nebo jsou zařazeny do front a čekají, až je ESB služba vyzvedne).
- Vzniká mnoho paralelních větví. Jejich vzájemná synchronizace je náročná. Příklad: Představme si, že máme 30 typů členství a každý typ má 200 000 kontraktů. 200 000 kontraktů musíme zpracovat po menších částech (jinak bychom měli příliš hrubou granularitu služeb). Pokud bychom kontrakty každého typu členství zpracovávali paralelně např. ve 20 větvích, pak by jedna větev zpracovávala 200 000 / 20 = 10 000 kontraktů. Dohromady nám v systému běží 30 * 20 = 600 větví paralelních větví výpočtu, což je zbytečně mnoho. Každý počítač v clusteru efektivně zpracovává omezené množství paralelních větví výpočtu (závislé na počtu procesorů a počtu jejich jader).
- Současné načítání identifikátorů všech typů členství zahltí databázi. Typů členství je přibližně 30. Některé typy mají přes 500 000 kontraktů.
- Paměťově náročné, protože se zpracovává všechno současně. V ESB službě, ve které se zpracovává dávka identifikátorů, se načítají jednotlivé záznamy kontraktů. Tyto záznamy jsou velké a jejich hodně. Souhrnná velikost všech kontraktů zpracovávaných v *Member fees* je přes 8 GB.

9.2.1.2 Definice business procesu Member fees – sekvenční zpracování typů členství a paralelní zpracování všech kontraktů

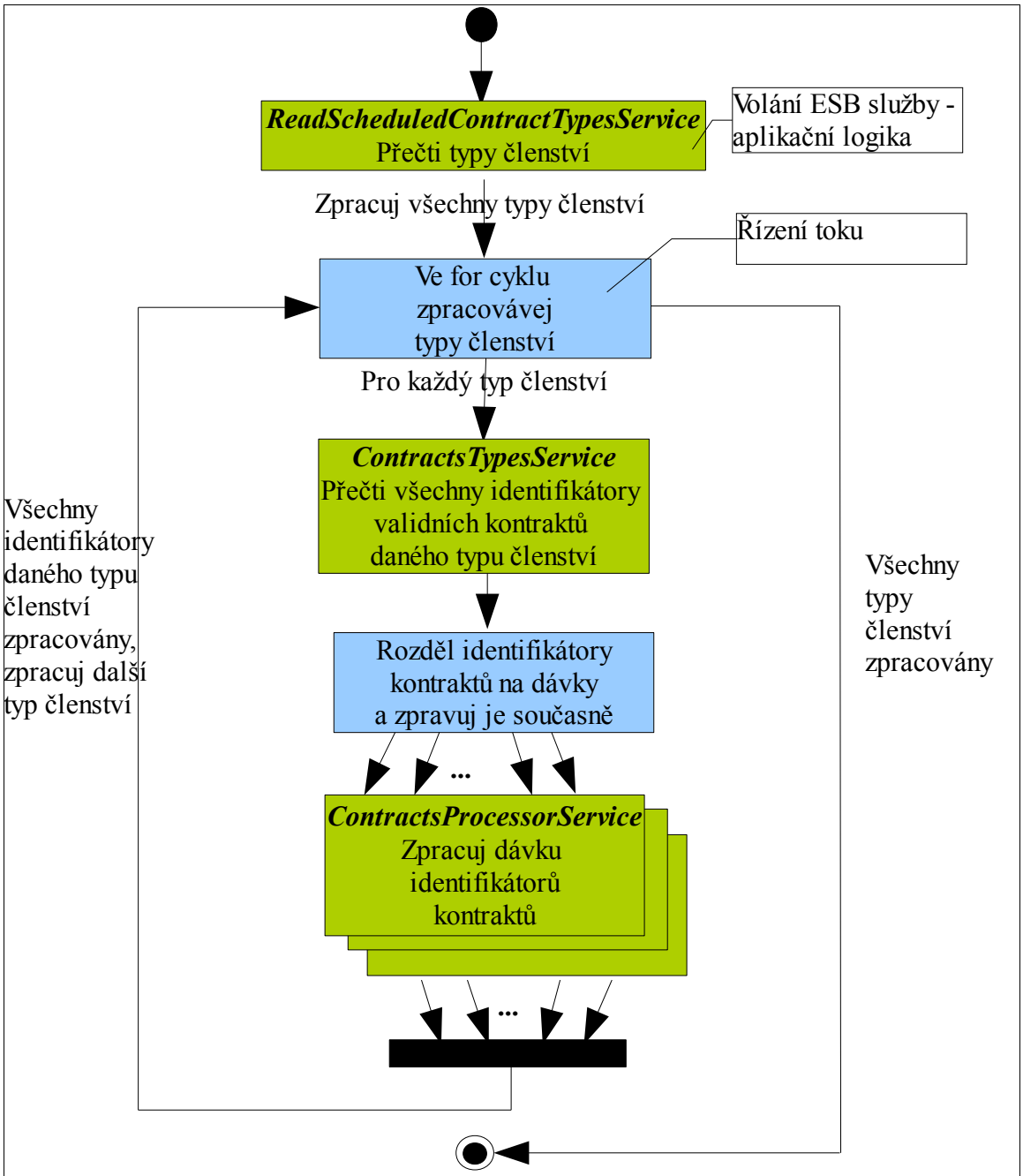
Druhý návrh (viz obrázek 14) zpracovává typy členství sekvenčně. Tím se snaží snížit počet paralelních větví výpočtu a omezit paměťovou náročnost. Identifikátory daného typu členství jsou rozděleny na dávky a tyto dávky jsou **paralelně** zpracovány.

Výhody návrhu:

- Návrh je jednoduchý a srozumitelný.
- Počet paralelních větví se dá jednoduše omezit zvětšením velikosti dávek.

Nevýhody návrhu:

- Takto navržený business proces je paměťově náročný. Některé typy členství mohou mít v budoucnosti obrovské množství kontraktů. V tomto návrhu se zpracovává celý typ členství najednou. Hrozí zde stejný problém jako v prvním návrhu – všechny kontrakty jednoho typu členství se nemusí vejít do paměti.
- Business proces je možné pozastavit, ale pozastavování trvá dlouho. Pozastavení fakticky proběhne až po zpracování rozpracovaného typu členství. Doba trvání zpracovávání jednoho typu členství může být několik desítek minut.

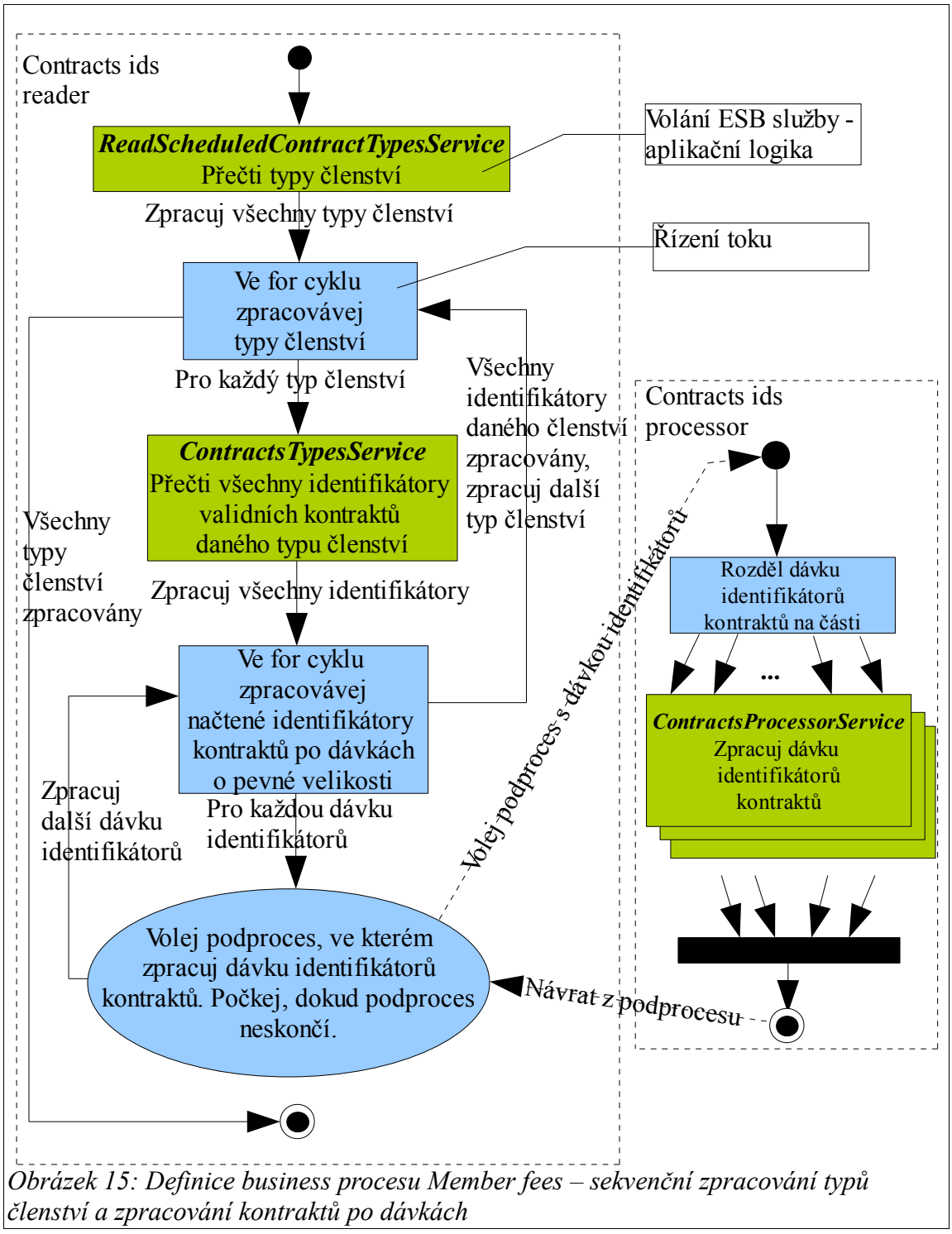


Obrázek 14: Definice business procesu Member fees – sekvenční zpracování typů členství a paralelní zpracování všech kontraktů

9.2.1.3 Definice business procesu Member fees – sekvenční zpracování typů členství a zpracování kontraktů po dávkách

Třetí návrh zobrazený na obrázku 15 vychází z předchozího návrhu. Jeho cílem je:

- další snížení paměťové náročnosti,
- zkrácení doby, po kterou trvá pozastavování procesu.



Obrázek 15: Definice business procesu Member fees – sekvenční zpracování typů členství a zpracování kontraktů po dávkách

V tomto návrhu jsou zpracovávány typy členství sekvenčně. Identifikátory kontraktů daného typu členství jsou zpracovávány sekvenčně po dávkách. Samotná dávka je rozsekána na části, které se zpracovávají paralelně. Tímto mírným zesložitěním návrhu se častěji vrací řízení procesnímu manažeru. Procesní manažer dokáže rychleji zareagovat na požadavek pozastavení business procesu. Ani v tomto návrhu není pozastavení okamžité. Doba trvání pozastavování procesu odpovídá době trvání

zpracovávání jedné dávky. Záleží na architektovi BP, jakou nastaví velikost dávek. Autor práce testoval proces *Member fees* při velikosti dávky 50 000 identifikátorů. Délka zpracování takovéto dávky je závislá na počtu počítačů zapojených do podnikové sběrnice služeb. V případě dvou počítačů trvá zpracování takovéto dávky přibližně 30 sekund.

Celý proces *Member fees* je dekomponován do dvou definic procesů:

- **Contracts ids reader**: Tento proces čte všechny validní identifikátory kontraktů (čte pouze identifikátory, protože všechny kontrakty daného typu členství by se nevešly do paměti) jednotlivých typů členství. Načtené identifikátory zpracovává v cyklu po dávkách o pevné velikosti. Sekvenčně volá podproces *Contracts ids processor*, kterému jako parametr předává dávku identifikátorů kontraktů. *Contracts ids reader* je dlouhotrvající proces (délka trvání v řádu desítek minut).

- **Contracts ids processor**: Podproces procesu *Contracts ids reader*. Proces *Contracts ids processor* zpracovává dávku identifikátorů kontraktů v několika paralelních větvích výpočtu. Vytváří Club transakce, které ukládá do databáze. Tento proces trvá relativně krátce (délka trvání v řádu desítek sekund).

Celý proces *Member fees* by mohl být zapsán jako jeden proces, ale není tak navržen z následujících důvodů:

1. Čím menší procesy jsou, tím jsou čitelnější a přehlednější.

2. Proces obsahující více paralelních větví výpočtu prakticky není možné pozastavit. Důvodem jsou technická omezení (více viz příloha *Technologická dokumentace, kapitola Implementace pozastavení instance procesu a omezení z toho vyplývající*) daná implementací procesního manažeru. Proto vzniklo rozdělení na dva procesy: nadřazený řídící proces s jednou výpočetní linií, a podproces s mnoha výpočetními liniemi, který je opakovaně volán. Pozastavován je pouze nadřazený řídící proces a podproces se nechává doběhnout.

Výhody návrhu:

- Business proces je možné pozastavit. Čím menší velikost dávky zvolíme, tím častěji budeme interagovat s procesním manažerem a tím dříve bude proces pozastaven. Na druhou stranu jakákoli interakce s procesním manažerem výpočet zpomaluje.

- Zpracovávání identifikátorů po dávkách snižuje paměťovou náročnost. Při zpracování další dávky můžeme uvolnit paměť, která byla použita při zpracování předchozí dávky (Club transakce lze odeslat do databáze a načtené kontrakty lze z paměti uvolnit).

- Je možné konfigurovat velikost dávek. Dávka se dále rozseká na části, které jsou zpracovávány paralelně. Počet těchto částí určuje míru paralelity.

Nevýhody návrhu:

- Požadavek na pozastavení business procesu není uspokojen okamžitě.
- Definice procesu se zkomplikovala. Není na první pohled zřejmé, co proces dělá.

9.2.1.4 Zvolená definice business procesu Member fees

Proces *Member fees* byl namodelován pomocí nástroje jBPM GPD [33] podle návrhu popsaného v kapitole 9.2.1.3. Bližší popis návrhu business procesu viz příloha *Dokumentace k business procesu Member fees*. Typy členství jsou zpracovávány sekvenčně. Načtené identifikátory kontraktů jsou rozděleny na dávky a dávky jsou zpracovávány jedna po druhé. Samotná dávka je rozdělena na části a tyto části jsou zpracovávány současně.

9.3 Implementace jednotlivých částí algoritmu

Většina logiky business procesu *Member fees* byla převzata ze systému Club. Byly znovupoužity především DAO [8] beany a DTO [10] beany. Logika implementující danou část algoritmu musela být přepsána do nových session bean.

9.4 Integrace business logiky do služeb

Služba volá session beany obsahující business logiku. Služba získá referenci na session beanu pomocí JNDI [37]. Protože se služba a session beana nalézají ve stejném adresovém prostoru, volání je velmi rychlé.

9.5 Zvolená verze distribuovaných transakcí

Ačkoliv *Member fees* je dlouhotrvající business proces, autor se rozhodl využít standardu WS-AtomicTransaction. Důvody pro to jsou:

- *Member fees* proces používá distribuovanou transakci pouze pro ukládání nových záznamů do databáze. Logika business procesu *Member fees* používá metodu optimistického zamykání, která používá zámky na jen nezbytně dlouhou dobu (více viz kapitola 8.2.4). Navíc business proces *Member fees* žádné záznamy nemění ani nemaže, takže zámky neblokují žádné jiné procesy v systému.
- WS-BusinessActivity vyžaduje složitější aplikační logiku vzhledem k nutnosti programovat kompenzační logiku.
- Kompenzační transakce se nemusí povést, což by bylo nepřípustné.

9.6 Shrnutí vlastností návrhu business procesu Member fees

Vlastnosti business procesu *Member fees*:

- **Celý běží v rámci jedné distribuované transakce**: Business proces *Member*

fees běží v rámci jedné distribuované transakce ve smyslu standardu WS-ActivityTransaction. Na začátku procesu distribuovaná transakce vznikne a na konci business procesu je transakce ukončena. Pokud se během vykonávání business procesu vyskytne chyba, transakce je odvolána. Tím jsou všechny změny, které logika business procesu způsobila v databázi, vráceny zpět.

- **Zpracovává typy členství sekvenčně**: Nejdříve proces zpracuje jeden typ členství, pak další atd.
- **Zpracovává identifikátory kontraktů po dávkách**: Identifikátory kontraktů daného typu členství zpracovává po dávkách. Dávka identifikátorů je pak rozdrobena na části, které jsou zpracovány současně.
- **Business proces je možné pozastavit a pak opětovně rozběhnout**: Proces je možné z rozhraní administrátora BP pozastavit a pak opětovně rozběhnout. Proces ale není pozastaven okamžitě. Průběh zpracovávání dávky identifikátorů totiž není možné pozastavit. Až se dávka zpracuje, proces se automaticky pozastaví.
- **Business proces je možné předčasně ukončit**: Proces je možné z rozhraní administrátora BP předčasně ukončit, a tím odvolat všechny změny, které logika business procesu provedla.

10 Analýza rychlosti zpracování BP *Member fees*

Celkovým vstupem business procesu *Member fees* je přibližně 2 000 000 kontraktů. Výstupem je přibližně 800 000 Club transakcí uložených do databáze. Z každého kontraktu vznikne maximálně jedna Club transakce. Z nezpoplatněných kontraktů Club transakce nevznikají.

Rychlost zpracování procesu *Member fees* autor analyzoval z hlediska vlivu velikosti clusteru.

Tabulka 2 popisuje počítače, které byly použity pro testování procesu *Member fees*.

Název počítače	CPU	RAM	Z toho RAM k dispozici pro JBoss
A	2x Intel Xeon 3,6 Ghz	7,1 GB	3 GB
B	2x Intel Xeon 3,4 Ghz	7,1 GB	3 GB
C	Intel Core 2 Duo 2,2 Ghz	2,0 GB	0,8 GB
OracleDB	2x Intel Xeon 3,6 Ghz	7,1 GB	N/A

Tabulka 2: Počítače použité pro účely testování

V průběhu vytváření statistik nebyly počítače A, B, C a OralceDB zatíženy žádnou jinou významnou činností. Počítače A, B, C jsou servery. Počítač C je notebook. Testování probíhalo v prostředí rychlé lokální sítě.

Tabulka 3 sumarizuje počty instancí služeb na jednotlivých počítačích.

Služba / Počítač	A	B	C
ReadScheduledContractTypesService	1	1	1
ContractsTypesService	1	1	1
ContractsProcessorService	10	10	10

Tabulka 3: Počet instancí služby na jeden počítač

Autor testoval proces *Member fees* na několika konfiguracích infrastruktury (co se týče počtu počítačů v infrastruktuře). Všechny testované konfigurace využívají stejný databázový server OracleDB.

Testované konfigurace:

1. samotný server A,
2. cluster obsahující servery A, B,
3. cluster obsahující servery A, B, C.

10.1.1 Testovaná konfigurace 1

Tato konfigurace obsahuje pouze server A a databázový server OracleDB.

Podproces *Contracts ids processor* (viz návrh BP *Member fees* v kapitole 9.2.1.3) dávku identifikátorů o velikosti 50 000 rozděluje na části o velikosti maximálně 5 000, aby vzniklo 10 paralelních větví výpočtu (10 paralelních větví výpočtu na jeden počítač).

Délka trvání procesu *Member fees*: **1 hodina 2 minuty 19 sekund**

10.1.2 Testovaná konfigurace 2

Tato konfigurace obsahuje server A, B a databázový server OracleDB. Počítače A a B jsou srovnatelně výkonné.

Na serveru A i serveru B byly aktivovány instance procesního manažeru za účelem zvýšení výkonu (více viz kapitola 7.2.1).

Podproces *Contracts ids processor* dávku identifikátorů o velikosti 100 000 rozděluje na části o velikosti maximálně 5 000, aby vzniklo 20 paralelních větví výpočtu (10 paralelních větví výpočtu na jeden počítač).

Délka trvání procesu *Member fees*: ***36 minut***

10.1.3 Testovaná konfigurace 3

Tato konfigurace obsahuje servery A, B, databázový server OracleDB a daleko slabší počítač C. Zde se významně projevila politika vyvažování zátěže, která rozděluje práci jednotlivým počítačům rovnoměrně. Počítače A a B většinu času čekaly, až počítač C svou práci dokončí.

Na serveru A, serveru B byly aktivovány instance procesního manažeru za účelem zvýšení výkonu (více viz kapitola 7.2.1). Na počítači C procesní manažer zapnut nebyl, aby nebyl slabší počítač C příliš zahlcen.

Podproces *Contracts ids processor* dávku identifikátorů o velikosti 150 000 rozděluje na části o velikosti maximálně 5 000, aby vzniklo 30 paralelních větví výpočtu (10 paralelních větví výpočtu na jeden počítač).

Délka trvání procesu *Member fees*: ***46 minut 14 sekund***

10.1.4 Vliv databáze systému Club na rychlost zpracování

Ukázalo se, že testovací databáze systému Club zvládá uspokojovat paralelní požadavky z mnoha instancí služeb. Business logika se na dvou počítačích prováděla přibližně dvakrát rychleji než na jednom počítači.

10.1.5 Vliv procesního manažeru na rychlost zpracování

Procesní manažer velmi zpomaluje celý průběh business procesu. A to i přesto, že je nasazeno více instancí procesních manažerů na různé uzly podnikové sběrnice. Při každé konfiguraci clusteru režie procesního manažeru činila přibližně 9 minut z celkové doby trvání zpracování BP. Příčinami jsou:

- **Přístup procesního manažeru k databázi**: Procesní manažer z databáze čte především kontext procesní instance. Aktualizovaný kontext instance pak zpětně zapisuje do databáze. Uložení kontextu je náročná operace, protože kontext instance může být velký – může obsahovat velké množství proměnných. Hodnoty proměnných se před zápisem do databáze musí serializovat. Podobně při čtení z databáze se musí hodnoty deserializovat. Serializace a deserializace jsou časově náročné operace.

- **Synchronizace jednotlivých paralelních větví procesu**: Procesní manažer používá metodu optimistického zamykání (o optimistickém zamykáni více viz kapitola 8.2.4) pro řešení konkurenčního přístupu k datům. Při synchronizaci paralelních větví v řídícím uzlu Join dochází v případě užití více instancí procesních manažerů k opakovanému vzniku kolizí. Vývojáři projektu jBPM uvažují o tom, že by procesní manažer neměl na relevantních tabulkách používat optimistické zamykání, ale na místo něj pesimistické zamykání, což by v případě zvýšené pravděpodobnosti vzniku kolize mělo vést ke zvýšení výkonu [62].

10.1.6 Shrnutí výsledků analýzy

Tabulka 4 shrnuje výsledky měření délky trvání business procesu *Member fees* při různých konfiguracích clusteru.

Číslo Konfigurace	**Počítače zahrnuté do clusteru**			**Délka trvání business procesu *Member fees***	**Z toho čas strávený v procesním manažeru**
	A	**B**	**C**		
1	✓			1 hod 2 min 19 sec	cca 9 min
2	✓	✓		36 minut 5 sec	cca 9 min
3	✓	✓	✓	46 min 14 sec	cca 9 min

Tabulka 4: Souhrnné výsledky – délka trvání Member fees

- Dva počítače zrychlily zpracování přibližně 1,7 krát oproti zpracování na jednom počítači. Většímu zrychlení brání vysoká režie procesního manažeru, kde se stráví přibližně 9 minut z celkové doby zpracování BP.

- Do navržené infrastruktury by se ve stávající implementaci měly dávat srovnatelně výkonné počítače. Vyvažování zátěže nebere ohled na výkonost jednotlivých počítačů – všem počítačům je rozdělována práce rovnoměrně. Následkem toho výkonnější počítače čekají na slabší.

11 Klíčové faktory mající vliv na zrychlení zpracování BP *Member fees*

Na zrychlení zpracování business procesu *Member fees* mají zásadní vliv následující faktory:

- snížení paměťové náročnosti business procesu,
- paralelní a distribuované zpracování business procesu,
- optimalizovaný přístup k databázi:
  - volba vhodné strategie uzamykání databáze,
  - odstranění zbytečných indexů,
  - JDBC batching,
  - využívání vyrovnávací paměti pro opakovaně čtená data.

11.1 Snížení paměťové náročnosti

Business proces *Member fees* zpracovává velké množství záznamů – přibližně dva milióny kontraktů. V logice *Member fees* byla chyba, která způsobovala, že business proces postupně načítal z databáze kontrakty, ale poté co je zpracoval, je neuvolňoval z paměti. Všechny kontrakty najednou se ale nevešly do 8 GB operační paměti produkčního serveru. Proto muselo být systému Club přiděleno více paměti, než kolik byla skutečná operační paměť. V momentě, kdy systém Club zaplnil celou operační paměť, operační systém začal odkládat stránky na disk (angl. *swapping* [63]). Tím se výkon celého serveru snížil na minimum.

Příčina takto vysokého nároku na paměť nebyla vůbec zřejmá. Ve firmě převládlo přesvědčení, že důvodem není chyba, ale že je to vlastnost business procesu *Member fees*.

Stejná chyba se nacházela i v nové implementaci *Member fees* nasazené na ESB, protože business logika procesu *Member fees* byla znovupoužita i s chybou. Autorovi diplomové práce se společně zaměstnancem společnosti Flynet podařilo tuto „zákeřnou“ chybu odhalit. Chyba spočívala v nesprávném používání projektu Hibernate [64], který umožňuje objektový přístup k relační databázi. K nalezení příčiny musel být použit program pro diagnostiku paměti virtuálního stroje Javy. Opravením chyby se snížila paměťová náročnost business procesu *Member fees* přibližně na 2 GB.

Okamžitým efektem bylo zkrácení doby zpracování business procesu *Member fees* o 13 hodin z původních 20 hodin.

11.2 Paralelní a distribuované zpracování business procesu

Požadujeme především co největší výkon ve zpracování business procesu. Abychom dosáhli největšího výkonu, musíme balancovat mezi velikostí zpracovávaných úkolů a komunikační režií.

Jemná granularita znamená, že individuální úkoly jsou relativně malé (velikost kódu, délka zpracování). Data jsou často přenášena mezi jednotlivými uzly systému [65].

Hrubá granularita znamená pravý opak: relativně velké úkoly, komunikace mezi uzly systému je méně častá [65].

Čím jemnější je granularita, tím většího paralelismu můžeme dosáhnout, ale musíme čelit větší režii na synchronizaci a komunikaci.

Příliš hrubá granularita vede k následujícím okolnostem:

- Infrastruktura pomalu reaguje na změnu (např. přidání dalšího počítače do infrastruktury).
- Na nějaké počítače nemusí zbýt práce a nevyužívá se tak jejich výkon.

Algoritmus business procesu *Member fees* je jednoduše paralelizovatelný. Na vstupu dostane velký balík identifikátorů kontraktů a každý kontrakt zpracuje. Je důležité, že zpracování jednoho kontraktu je **nezávislá** operace. Algoritmus by se dal zobecnit takto: algoritmus na vstupu dostane množinu prvků a každý tento prvek nezávisle zpracuje. Podobné schéma má v dnešním podnikání hodně procesů:

- zpracování faktur,
- přičítání úroků k účtům,
- výpočet poplatků za služby,
- ...

Vstupem *Member fees* je přibližně 2 000 000 identifikátorů kontraktů. Nemůžeme ale zpracovávat kontrakty po jednom. To by způsobilo příliš velkou komunikační režii. Vhodná volba je zpracovávat kontrakty seskupené do větší dávky o velikosti např. 5000 kontraktů. Dávky jsou zpracovány současně, ale obsah dávky je zpracován sekvenčně.

Začleněním více počítačů do procesu zpracování dosahujeme distribuovaného zpracování. Každý počítač začleněný do infrastruktury je schopen zpracovávat několik dávek současně. Tím dosahujeme i paralelního (v rámci jednoho počítače [1]) zpracování.

Analýza rychlosti zpracování BP *Member fees* vzhledem k velikosti clusteru je provedena v kapitole 10.

11.3 Optimalizovaný přístup k databázi

11.3.1 Volba vhodné strategie uzamykání databáze

V celém systému Club se používá optimistické zamykání (více viz kapitola 8.2.4), protože se předpokládá, že ke konfliktům bude docházet velmi zřídka. U business procesu *Member fees* je také aktivováno optimistické zamykání. Je na aplikačním programátorovi, jak bude reagovat na vzniklou kolizi. Používaná praktika je pokusit se

odvolanou transakci provést znova.

Business proces *Member fees* žádné záznamy nemění, proces *Member fees* pouze ukládá nové záznamy. Není důvod, proč u procesu *Member fees* používat pesimistické zamykání.

11.3.2 Odstranění zbytečných indexů

Business proces *Member fees* ukládá do tabulky *transactions_* záznamy o zpoplatnění kontraktů. Těchto záznamů tam celkem uloží přibližně 800 000. Během analýzy chování procesu *Member fees* se zjistilo, že samotné vložení 800 000 záznamů do této tabulky trvá přes čtyři hodiny. Autor postupně vyloučil možné příčiny – přetížení sítě, pomalost disků. Po několika dnech hledání příčiny se autorovi podařilo najít pravý důvod takového zpoždění – indexy na tabulce *transactions_*.

Na tabulce *transactions_* se nacházelo množství indexů. Vypnutí všech indexů okamžitě snížilo dobu vkládání na 15 minut. Cílem analýzy indexů na tabulce *transactions_* bylo zjistit, které indexy mají největší vliv na zpomalení vkládání záznamů.

Tabulka 5 ukazuje indexy, které jsou definované na tabulce *transactions_*.

Číslo indexu	**Index na sloupce (Oracle Datatypes [66])**	**Poznámka**
1	NUMBER	Index na primární klíč
2	NUMBER, NUMBER	
3	VARCHAR2(255), TIMESTAMP	Doména sloupce typu VARCHAR2 je v tomto případě obrovská
4	NUMBER, TIMESTAMP, NUMBER	
5	VARCHAR2(255)	Doména sloupce typu VARCHAR2 je v tomto případě několik málo hodnot
6	NUMBER, VARCHAR2(255), TIMESTAMP	Doména sloupce VARCHAR2 je v tomto případě obrovská

*Tabulka 5: Indexy na tabulce transactions_*

Průběh testu: V tabulce *transactions_* byla aktivována jistá podmnožina indexů a sledovalo se, jak dlouho trvá vložení vložení 50 000 záznamů v jedné transakci. Tabulka již obsahovala asi 2 000 000 záznamů. Měření vždy probíhalo třikrát.

Čísla indexů – označeno, pokud je index aktivovaný						Doba, za jak dlouho se podařilo vložit 50 000 záznamů do tabulky *transactions_* obsahující dva milióny řádků		
1	**2**	**3**	**4**	**5**	**6**	**Čas 1**	**Čas 2**	**Čas 3**
✓						1 min 10 sec	50 sec	46 sec
✓	✓					56 sec	55 sec	56 ses
✓		✓				7 min 30 sec	8 min 42 sec	7 min 49 sec
✓			✓			1 min 44 sec	1 min 36 sec	1 min 33 sec
✓				✓		51 sec	48 sec	52 sec
✓					✓	3 min 29 sec	5 min 8 sec	4 min 33 sec
✓	✓	✓	✓	✓	✓	13 min 48 sec	14 min 30 sec	13 min 22 sec

Tabulka 6: Testování doby vkládání záznamů do tabulky transactions_ při různé konfiguraci aktivovaných indexů

Závěr testu: Autor práce ukázal, že největší zpomalení vkládání způsobují indexy na sloupce VARCHAR2(255), které mají velkou doménu. Index na sloupec typu TIMESTAMP také znatelně zpomaluje vkládání. Na základě provedených testů zaměstnanci společnosti Flynet „špatné“ indexy 3 a 6 vypnuli, protože se ukázalo, že je již žádná část systému nevyužívá. Vypnutím těchto indexů se dosáhlo přibližně třináctinásobného zrychlení vkládání záznamů do tabulky *transactions_*.

11.3.3 JDBC batching

Jedna z pokročilých vlastností JDBC 2.0 [67] je schopnost seskupit několik SQL příkazů a poslat je do databáze jako jednu dávku. JBDC batching vede ke zvýšení výkonu a ke zmenšení počtu příkazů posílaných do databáze.

V testech se ukázalo, že je velký rozdíl mezi velikostí dávky 1 a velikostí dávky např. 15. Příliš nízká velikost dávky může vést k zahlcení databáze SQL příkazy. Zvětšení velikosti dávky na 15 vedlo k 10% zvýšení výkonu. Velikost dávky by se měla pohybovat mezi 10 až 50 [68]. Velké velikosti dávek mohou vést k problémům s pamětí [69].

11.3.4 Využívání vyrovnávací paměti pro opakovaně čtená data

I když dotaz směřuje do relativně malé tabulky (např. 50 řádků), odezva databáze je výrazně horší (10krát a více – podle složitosti vyhodnocení dotazu a míry zahlcení databáze) než odezva vyrovnávací paměti.

Existuje celá řada druhů vyrovnávacích pamětí. Data určená pouze pro čtení je nejefektivnější ukládat v lokální (nedistribuované) read-only vyrovnávací paměti.

V případě, že je nutné čtená data modifikovat, musí se použít jiný typ vyrovnávací paměti. Podle požadavků lze v prostředí J2EE vybírat mezi read-write, nonstrict read/write a transactional cache [70].

11.4 Souhrn přínosu optimalizací a nasazení ESB

Proces *Member fees* byl zrychlován postupně aplikováním optimalizací. Bez optimalizací trval business proces *Member fees* na **jednom počítači** v testovacím i produkčním prostředí přibližně 20 hodin. Aplikováním všech optimalizací se podařilo zrychlit proces *Member fees* v testovacím prostředí (produkční prostředí viz kapitola 11.4.1) z 20 hodin na 1 hodinu. Tabulka 7 ukazuje přínos jednotlivých optimalizací (měřeno na stejném vzorku dat – zpracování přibližně 2 000 000 kontraktů).

Optimalizace	**Zkrácení délky trvání business procesu *Member fees* z 20 hodin**
Snížení paměťové náročnosti	přibližně o 13 hodin
JDBC batching, odstranění zbytečných indexů	přibližně o 4 hodiny
Využívání vyrovnávací paměti pro opakovaně čtená data	přibližně o 2 hodiny

Tabulka 7: Přínos jednotlivých optimalizací na délku trvání business procesu Member fees

Použití podnikové sběrnice služeb umožnilo další zrychlování procesu. Nasazení dvou počítačů do ESB umožnilo již optimalizovaný business proces *Member fees* ještě zrychlit přibližně 1,7krát – z jedné hodiny na 36 minut.

Souhrnně se povedlo zrychlit business proces *Member fees* (v testovacím prostředí) z 20 hodin na 36 minut, což je přibližně 33krát rychlejší zpracování.

11.4.1 Business proces *Member fees* v produkčním prostředí

V současnosti se v produkčním systému, který běží na jednom velmi výkonném serveru, nachází optimalizovaná implementace business procesu *Member fees* **bez** podnikové sběrnice služeb. Ačkoli produkční systém obsahuje podobné množství dat jako testovací prostředí, zpracování business procesu *Member fees* trvá na produkčním serveru přibližně 3 hodiny. Je to způsobené tím, že na produkčním serveru běží mnoho dalších úloh a jiných business procesů, které odčerpávají serveru výkon. V produkčním systému by tak daleko více vynikla výhoda nasazení podnikové sběrnice služeb než v testovacím prostředí. Vzhledem k provedeným testům autor diplomové práce očekává, že při nasazení dvou počítačů by se délka trvání *Member fees* zkrátila minimálně na 1,5 hodiny. Zrychlení by bylo pravděpodobně ještě větší, protože zátěž by se rozložila na více počítačů. Navíc distribuovaný systém může mít dohromady mnohem více přidělené operační paměti.

12 Průběh vývoje distribuované infrastruktury

Autor používal vývojové prostředí Eclipse [34]. Eclipse byl vybrán především pro množství rozšíření (pluginů), které do něj je možné nainstalovat. Aplikační server JBoss je pomocí takového rozšíření také s prostředím Eclipse integrován. Díky integraci JBoss a prostředí Eclipse je možné spustit aplikační server v debug módu a tím snadněji hledat chyby ve vyvíjených aplikacích.

Testování nové funkcionality probíhalo na podnikové sběrnici nasazené na osobním notebooku. Až poté, co byla daná funkcionalita odladěna a otestována na jednom počítači, byla testována na clusteru.

K testování měl autor k dispozici vlastní notebook a stolní počítač společnosti Flynet. Dále společnost Flynet poskytla dva výkonné servery a dva testovací databázové stroje. První databázový stroj, který obsahuje velmi málo dat systému Club, slouží k ladění a testování funkcionality. Druhý databázový stroj obsahuje podobné množství dat systému Club jako produkční databáze. To umožnilo vytvořit srovnatelné prostředí jako má produkční systém.

12.1 Použité technologie a software

V následujících podkapitolách jsou vyjmenovány nejdůležitější použité technologie a software. Jsou členěny podle systémů, ve kterých se vyskytují. V části 'Ostatní' jsou uvedeny nástroje a software užitý pro vývoj.

12.1.1 Aplikační server

- JBoss AS 4.2.2 – aplikační server, na který se nasazuje celý distribuovaný framework (ESB, procesní manažer) a business logika [71]
- Hibernate 3 – implementace JPA [64]
- JDBC – propojení databáze a Javy [67]
- EJB 3.0 – vrstva aplikační logiky [5]
- Oracle 10g Enterprise Edition – databáze systému Club [72]
- Oracle XE – testovací databáze [72]
- Java SE 5 – virtuální stroj, základní sada knihoven [73]
- J2EE 1.4 - prostředí aplikačního serveru JBoss [74]

12.1.2 Distribuovaný framework

- JBossMQ – poskytovatel JMS [75]
- JBossESB – podniková sběrnice služeb [23]
- JBoss jBPM – procesní manažer [29]

- JBoss Transaction – implementace JTA transakcí a rodiny protokolů WS-Transaction [76]
- XML technologie (XML, DTD, XML Schema) – konfigurační soubory [77]
- Log4j – logování [78]
- Web Services – koordinace mezi koordinátorem a účastníkem distribuované transakce [79]

12.1.3 Ostatní

- Eclipse – vývojové prostředí [34]
- JBoss Tools – sada rozšíření pro Eclipse, která do vývojového prostředí integruje podporu pro aplikační server JBoss [80]
- jBPM GPD – sada rozšíření pro Eclipse, která umožňuje vytvářet a editovat komplexní business procesy napsané v jazyce jPDL [33].
- Ant – build projektu [81]
- Oracle SQL Developer – vývoj SQL dotazů [72]
- SUSE Linux Enterprise Server - testovací operační systém [82]
- Fedora 8 - testovací operační systém [83]
- Windows Vista – testovací operační systém [84]

13 Zhodnocení splnění cílů, slabiny, problémy při vývoji a budoucí vývoj

13.1 Zhodnocení splnění cílů

Všechny cíle vytyčené v kapitole 3.2 se podařilo splnit až na transparentní spojení původního a nového systému z pohledu administrátora BP. Původní systém a vytvořená infrastruktura vystavují rozdílná webová rozhraní. V budoucím vývoji je nutné webová rozhraní sjednotit.

Cíl první: Vytvoření škálovatelné distribuované infrastruktury

Cíl se podařilo splnit nasazením podnikové sběrnice služeb. Podniková sběrnice poskytuje potřebné funkce pro distribuované zpracování business procesů.

Začlenění více počítačů do procesu zpracování je z pohledu aplikačního programátora transparentní.

Jako middleware bylo zvoleno posílání JMS zpráv. Výhodami JMS jsou především:

- asynchronnost a spolehlivost komunikace,
- možnost začlenit poslání zprávy do transakce.

Cíl druhý: Maximální využití paralelity algoritmu BP

Je v plné moci architekta business procesů, aby navrhl vhodnou granularitu služeb. Podniková sběrnice umožňuje nasadit více instancí služeb na jednotlivé počítače zapojené do podnikové sběrnice a tím využít výkonu všech procesorů a procesorových jader.

Cíl třetí: Zrychlení zpracování BP a zefektivnění využívání hardwarových prostředků (především procesorového výkonu a paměti)

Autorovi se podařilo najít největší nehospodárnosti ve zpracování podnikového procesu *Member fees*. Na zrychlení zpracování business procesu mají zásadní vliv následující faktory:

- snížení paměťové náročnosti,
- odstranění některých indexů, které velmi zpomalovaly vkládání nových záznamů,
- nasazení vyrovnávací paměti na opakovaně čtená data,
- používání JDBC batching (seskupení SQL příkazů do dávek).

Tabulka 8 ukazuje přínos jednotlivých optimalizací na zrychlení business procesu *Member fees*, jehož zpracování na jednom počítači bez optimalizací trvalo přibližně 20 hodin.

Optimalizace	Zkrácení délky trvání business procesu *Member fees* z 20 hodin
Snížení paměťové náročnosti	přibližně o 13 hodin
JDBC batching, odstranění zbytečných indexů	přibližně o 4 hodiny
Využívání vyrovnávací paměti pro opakovaně čtená data	přibližně o 2 hodiny

Tabulka 8: Přínos jednotlivých optimalizací na délku trvání business procesu Member fees

Testy ukázaly, že dva počítače začleněné do podnikové sběrnice služeb zrychlí zpracování přibližně 1,7krát oproti zpracování na jednom počítači. Lineárnímu zrychlení brání především vysoká režie procesního manažeru.

Zátěž na procesory a paměť je distribuovaná na více počítačů. V důsledku to znamená úsporu peněz, protože není nutné kupovat drahý superpočítač. Několik počítačů střední třídy udělá stejnou práci jako jeden superpočítač.

Cíl čtvrtý: Kontrola a řízení BP

Administrátor BP používá administrátorské rozhraní, pomocí kterého může business procesy řídit a monitorovat. Rozhraní administrátora BP umožňuje:

- monitorovat stav business procesu,
- spustit BP,
- pozastavit BP (suspend),
- opětovně rozběhnout BP (resume),
- předčasně ukončit BP.

Cíl pátý: Vykonání BP v rámci jedné distribuované transakce

Do frameworku se podařilo integrovat implementaci standardů WS-Transaction. Jsou k dispozici dva koordinační protokoly: WS-AtomicTransaction a WS-BusinessActivity.

ESB služby, plnící roli iniciátora distribuované transakce, lze začlenit do návrhu business procesu v jBPM GPD. Činnost ESB služby může být asociována s globální transakcí resp. business aktivitou.

Cíl šestý: Oddělení komunikační logiky a business logiky

Komunikační logika a business logika je odděleny díky použití podnikové sběrnice služeb. Business logika je dekomponována do služeb. Procesní manažer služby koordinuje podle předem nadefinovaného plánu.

Aplikační programátor se koordinaci vůbec nestará. Úkolem aplikačního programátor je pouze naimplementovat business logiku jednotlivých služeb.

Cíl sedmý: Maximální znovupoužití implementace stávajících BP

JBossESB běží na aplikačním serveru JBoss. Aplikační logika všech business procesů Clubu je nasazena na aplikačním serveru JBoss. Tato shoda prostředí umožňuje, aby ESB služby přímo využívaly session bean business logiky systému Club – především těch bean, které přistupují k datům v databázi. Logika implementující danou část algoritmu musela být přepsána do nových session bean. Toto přepsání celkově trvalo přibližně 4 dny.

Cíl osmý: Postupný přechod ze starého systému na novou infrastrukturu

Plánuje se postupné nasazování dlouhotrvajících business procesů na novou infrastrukturu. Distribuovaný framework může být nasazen na stejný aplikační server, na kterém běží původní systém Club. Některé business procesy budou využívat distribuovaný framework, ostatní mohou využívat původní systém Club.

Cíl devátý: Transparentnost spojení původního a nového systému z pohledu administrátora BP

Původní systém a vytvořená infrastruktura vystavují pro administrátora BP rozdílná webová rozhraní. V budoucím vývoji je nutné webová rozhraní sjednotit, aby administrátor BP nemusel používat více webových rozhraní. Tento cíl nebyl dosud implementován a bude předmětem budoucího vývoje.

Cíl desátý: Monitoring infrastruktury

JBossESB poskytuje webové rozhraní, kterým lze u každé instance služby zejména sledovat:

- počet úspěšně, neúspěšně zpracovaných zpráv,
- celkovou délku trvání zpracování zprávy.

Naměřené hodnoty jsou přehledně v čase zobrazovány do grafu.

Z výpisů je možné zjistit výkonnost jednotlivých uzlů podnikové sběrnice. Dále je možné zjistit služby, které nejdéle zpracovávají vstupní zprávu. Monitorování provozu je důležité pro nacházení úzkých míst v návrhu BP.

Toto webové rozhraní bylo během vývoje velmi používáno, pomáhalo nacházet úzká místa řešení.

13.2 Slabiny současné implementace infrastruktury

Současná implementace infrastruktury má následující slabiny:

- **Neexistence modulu zotavení po pádu koordinátora nebo účastníka distribuované transakce**: V budoucím vývoji je nutné dodat do implementace standardů WS-Transaction modul zotavení po výpadku koordinátora nebo účastníka distribuované transakce. Více viz kapitola 8.4.

- **Nemožnost pokračovat v business procesu v případě pádu některého ze serverů**: Pokud počítač, který se účastní zpracování business procesu, zhavaruje, tak není možné pokračovat v daném business procesu. Administrátor BP musí business proces předčasně přerušit z administrátorského rozhraní. Možnost pokračovat v business procesu v případě pádu některého ze serverů bude předmětem budoucího vývoje. Dále viz kapitola 13.2.1.
- **Spravedlivé rozdělování práce na jednotlivé počítače**: Vyvažování zátěže by mělo reflektovat to, že některé počítače v clusteru jsou slabší a mělo by takovýmto počítačům přidělovat méně práce. Dále viz kapitola 13.2.2.

13.2.1 Nemožnost pokračovat v BP v případě pádu některého ze serverů

Zaměstnanci společnosti Flynet mají zkušenost, že v reálných podmínkách nehrozí ani tak pád serverů jako spíše nekonzistence dat v databázi nebo chyba v logice business procesu. Každé přerušení provádění je vnímáno jako kritická chyba a je nutný zásah administrátora. Následuje podrobné hledání příčiny chyby.

Díky diplomové práci se dosáhlo tak významného zrychlení business procesu *Member fees*, že je možné business proces v případě chyby spustit znova a zároveň neohrozit časový plán měsíční uzávěrky. Přesto by do budoucna bylo vhodné integrovat do infrastruktury funkci automatického zotavení po pádu některého ze serverů tak, aby bylo možné v rozpracovaných úkolech plynule pokračovat.

13.2.2 Spravedlivé rozdělování práce na jednotlivé počítače

Vytvořená infrastruktura byla šitá na míru pro prostředí, které je ve společnosti Flynet. Společnost Flynet vlastní podobně výkonné počítače. Do budoucna je ovšem nutné toto omezení odstranit.

Podniková sběrnice je nakonfigurována tak, aby používala politiku vyvažování zátěže RoundRobin (viz kapitola 7.2.2), která rozděluje práci „spravedlivě“. Tato politika nebere ohled na výkonnost jednotlivých počítačů v clusteru, což může vést k tomu, že výkonnější počítače čekají na slabší počítače. Možným řešením je použití clusterovaných front [85]. Clusterovaná fronta je „logická“ fronta rozprostřená na více uzlů v clusteru. Messaging systém zprávy automaticky přemisťuje na rychlejší uzly clusteru.

Projekt JBossMessaging [86], který implementuje JMS [13], clusterované fronty poskytuje. Původně byl tento projekt ve vytvářené infrastruktuře použit, ale množství chyb v jeho implementaci zabránilo jeho nasazení. Implementace JMS projektem JBossMQ [75] se ukázala jako spolehlivější. JBossMQ ovšem clusterované fronty neposkytuje.

Použití clusterovaných front v momentě, kdy bude projekt JBossMessaging stabilní, tuto slabinu odstraní. Je otázkou, do jaké míry bude mít nasazení clusterovaných front negativní vliv na výkonnost messaging systému.

13.3 Problémy při vývoji

Při vývoji distribuovaného frameworku nastalo mnoho komplikací. K vytvoření frameworku byly až na databázi Oracle použity open source projekty. Problémem open source projektů je jejich pochybná kvalita, nepravidelné nebo příliš rychlé releasy, malé množství dokumentace, nízká úroveň podpory starších releasů a další.

Dokumentace jednotlivých projektů použitých při vytváření frameworku tvrdí, že projekty mohou být nasazeny v clusteru počítačů. Realita je ovšem jiná. Většina takových produktů je otestovaných pouze na jednom počítači. Zevrubným testováním produktů v distribuovaném prostředí se vývojáři zabývají až na „druhém“ místě. Autor objevil velké množství chyb prakticky ve všech použitých projektech. Autor se vždy ve vlastním zájmu snažil najít a opravit příčinu chyby, aby mohl dál pokračovat v práci. Příčina a náprava byla poté ohlášena vývojářům, aby v příštím „stabilním“ releasu již tato chyba nebyla. Vedlejším produktem bylo, že se autor seznámil s implementací jednotlivých technologií.

Přechod na novější (méně chybovou) verzi projektu třetí strany znamená čelit problémům s nekompatibilitou s ostatními projekty.

Po čas vývoje autor změnil poskytovatele Java Message Service [13] (JMS) v podnikové sběrnici služeb. Autor měl v úmyslu přejít ze starší implementace JBossMQ [75] na novější implementaci standardu JMS projektem JBossMessaging [86]. JBossMessaging poskytuje funkce týkající se bezpečnosti posílání zpráv, clusterované fronty [85] aj. Množství bugů, které se objevilo při testování, však autora od použití JBossMessaging odradilo.

V neposlední řadě si autor problémy způsoboval sám. Nesprávné používání projektu Hibernate [64], který umožňuje objektový přístup k relační databázi, vedlo k únikům paměti (memory leak). K nalezení příčiny musel být použit program pro diagnostiku paměti virtuálního stroje Javy. Získané zkušenosti byly okamžitě předávány zaměstnancům společnosti Flynet. Následně se ukázalo, že podobné pochybení se vyskytuje i v jiných business procesech systému Club.

13.4 Budoucí vývoj systému

Hlavními úkoly do budoucna jsou:

- **Vyřešení slabin infrastruktury**: Je vhodné integrovat do infrastruktury vyvažování zátěže, které bere v úvahu výkonnost počítače, možnost pokračovat v business procesu v případě pádu některého ze serverů a automatické zotavení po pádu účastníka nebo koordinátora

- **Zavedení priorit business procesů**: V současné implementaci infrastruktury mají všechny nasazené business procesy stejnou prioritu. Není možné zpomalit jeden business proces, aby jiný mohl běžet rychleji. Autor se seznámil se zdrojovým kódem procesního manažeru. Je možné relativně jednoduše upravit zdrojový kód procesního manažeru tak, aby administrátorem určené business procesy byly preferovanější před jinými.

- **Jedno administrátorského rozhraní**: V současné době vytvořený framework

vystavuje vlastní administrátorské rozhraní. Společnost Flynet již dlouho provozuje vlastní portál, kterým administruje business procesy. Je nutné stávající podnikový portál rozšířit tak, aby umožnil ovládání business procesů nasazených v podnikové sběrnici služeb.

V sídle společnosti Flynet autor prezentoval výsledky své práce majitelům společnosti a jejich zahraničnímu partnerovi. Momentálně se čeká na vyjádření koncernu. Až poté bude možné začlenit podnikovou sběrnici služeb do produkčního systému a integrovat do ní další dlouhotrvající business procesy.

14 Vyjádření společnosti Flynet

Projekt pana Kadlece, který s námi spolupracuje téměř dva roky, splnil naše požadavky a očekávání.

Nalezení úzkých míst a významné zrychlení procesu Member-fees přineslo podstatné snížení časové náročnosti tohoto kritického procesu používaného při uzávěrkách měsíce.

Rostoucí objem dat nás nutí být připraven na situaci, kdy stávající řešení systému nebude schopno dokončit náročné procesy v časově akceptovatelném časovém úseku. A právě výsledky diplomové práce umožňují obejít toto kritické omezení a umožní systému splňovat i v budoucnu náročné požadavky našich zákazníků.

15 Závěr

Motivací diplomové práce bylo zrychlit vnitropodnikový proces *Member fees*, který původně trval 20 hodin, distribuovaným zpracováním. Proces *Member fees* se podařilo zrychlit z 20 hodin na 36 minut – 33krát. Klíčů pro tak výrazné zrychlení bylo více:

- optimalizace business logiky procesu *Member fees*:
  - snížení paměťové náročnosti,
  - odstranění databázových indexů,
  - nezahlcování databáze SQL dotazy,
  - využívání vyrovnávací paměti pro opakovaně čtená data,
- distribuované zpracování:
  - nasazení podnikové sběrnice služeb.

V této diplomové práci byl navržený škálovatelný framework, který umožňuje distribuované zpracování dávkových vnitropodnikových business procesů. Jeho základními kameny jsou:

- **Enterprise service bus**: ESB poskytuje služby middlewaru. Logika business procesu je dekomponována do služeb, které jsou připojeny do podnikové sběrnice. Služby mohou být vystaveny na jednom počítači ve více instancích, takže instance služeb zpracovávají více požadavků současně. Distribuované zpracování je zajištěno tím, že instance služeb mohou být vystaveny na mnoha počítačích.
- **Business proces management**: Procesní manažer řídí tok zpráv mezi službami podle předem nadefinovaného plánu. Programátor business logiky se o tok zpráv nemusí starat. Pomocí procesního manažeru je možné řídit průběh business procesu – pozastavit ho, opětovně ho rozběhnout nebo ho předčasně ukončit.
- **Sada standardů WS-Transaction**: Pomocí standardů WS-AtomicTransaction nebo WS-BusinessActivity je zajištěno, že business proces běží v jedné distribuované transakci.

16 Seznam použitých zkratek

2PC – Two Phase Commit Protocol

API – Application Programming Interface

ANT – Another Neat Tool

BP – Business proces

CASE – Computer Aided Software Engineering

CORBA – Common Object Request Broker Architecture

DAO – Data Access Object

DTO – Data Transfer Object

EAP – JBoss Enterprise Application Platform

EJB – Enterprise Java Beans

EMS – Enterprise Messaging systém

EPR – Endpoint Reference

ESB – Podniková sběrnice služeb (Enterprise Service Bus)

GPD – Graphical Process Designer

IIOP – Internet Inter-ORB Protocol

J2EE – Java 2 Platform, Enterprise Edition

JBPM – Java Business Process Management

JDBC – Java Database Connectivity

JCA – J2EE Connector Architecture

JMS – Java Message Service

JNDI – Java Naming and Directory Interface

JPA – Java Persistence API

JSR – Java Specification Request

JTS – Java Transaction Service

JVM – Java Virtual Machine

MDB – Message Driven Bean

OTS – Object Transaction Service

RAM – Random-Access Memory

SLSB – Stateless Session Bean

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

WS-AT – Web Services Atomic Transaction

WS-C – Web Services Coordination

WS-BA – Web Services Business Activity

WS-T – Web Service Transaction

WSDL – Web Service Definition Language

XML – Extensible Markup Language

17 Literatura

[1] Wikipedia: *Distributed computing*. http://en.wikipedia.org/wiki/Distributed_computing, 2008.

[2] Hammer, Michael and Champy, James: *Reengineering the Corporation: A Manifesto for Business Revolution,* Harper Business, 1993.

[3] Wikipedia: *Java Platform, Enterprise Edition*. http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition, 2008.

[4] Wikipedia: *Activity Diagram*. http://en.wikipedia.org/wiki/Activity_diagram, 2008.

[5] Java Community Process: *Enterprise JavaBeans 3.0, JSR 220.* http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html, 2006.

[6] Wikipedia: *Session Beans*. http://en.wikipedia.org/wiki/Session_Beans, 2008.

[7] Wikipedia: *Message Driven Bean*. http://en.wikipedia.org/wiki/Entity_Bean, 2008.

[8] Wikipedia: *Data Access Object*. http://en.wikipedia.org/wiki/Data_Access_Object, 2008.

[9] Wikipedia: *Entity Bean*. http://en.wikipedia.org/wiki/Entity_Bean, 2008.

[10] Wikipedia: *Data Transfer Object*. http://en.wikipedia.org/wiki/Data_Transfer_Object, 2008.

[11] Wikipedia: *Message Passing*. http://en.wikipedia.org/wiki/Message_passing, 2008.

[12] Hohpe Gregor, Woolf Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley, 2003.

[13] Sun Microsystems, Inc.: *Java Message Service (JMS)*. http://java.sun.com/products/jms/, 2002.

[14] Sun Microsystems, Inc.: *J2EE Connector Architecture*. http://java.sun.com/j2ee/connector/, 2003.

[15] Wikipedia: *Quality of service.* http://en.wikipedia.org/wiki/Quality_of_service, 2008.

[16] Wikipedia: *Event driven programming.* http://en.wikipedia.org/wiki/Event-driven, 2008.

[17] Wikipedia: *Enterprise JavaBean*. http://en.wikipedia.org/wiki/Enterprise_JavaBean, 2008.

[18] Sun Microsystems, Inc.: *Java Transaction Service (JTS)*. http://java.sun.com/javaee/technologies/jts/index.jsp, 1999.

[19] Wikipedia: *Enterprise service bus*. http://en.wikipedia.org/wiki/Enterprise_Service_Bus, 2008.

[20] Wikipedia: *Enterprise messaging system*.

http://en.wikipedia.org/wiki/Enterprise_messaging_system, 2008.

[21] World Wide Web Consortium (W3C): *SOAP*. http://www.w3.org/TR/soap12-part1/, 2007.

[22] InfoWorld: *Best of open source software awards: Platforms and middleware*. http://weblog.infoworld.com/tcdaily/archives/2008/08/best_of_open_so_4.html, 2008.

[23] Red Hat Middleware: *JBossESB*. http://www.jboss.org/jbossesb/, 2008.

[24] Enterprise Integration Patterns: *Message Endpoint*. http://www.enterpriseintegrationpatterns.com/MessageEndpoint.html, 2008.

[25] Enterprise Integration Patterns: *Pipes and Filters*. http://www.eaipatterns.com/PipesAndFilters.html, 2008.

[26] Wikipedia: *Orchestration (computers)*. http://en.wikipedia.org/wiki/Orchestration_(computers), 2008.

[27] Wikipedia: *Business Process Execution Language*. http://en.wikipedia.org/wiki/Business_Process_Execution_Language, 2008.

[28] World Wide Web Consortium (W3C): *WSDL*. http://www.w3.org/TR/wsdl, 2001.

[29] Red Hat Middleware: *jBPM*. http://www.jboss.org/jbossjbpm/, 2008.

[30] Wikipedia: *Workflow engine*. http://en.wikipedia.org/wiki/Workflow_engine, 2008.

[31] Red Hat Middleware: *jBPM Process Definition Language (JPDL)*. http://docs.jboss.org/jbpm/v3/userguide/jpdl.html, 2008.

[32] Wikipadia: *Workflow*. http://en.wikipedia.org/wiki/Workflow, 2008.

[33] Red Hat Middleware: *jBPM GPD*. http://www.jboss.org/jbossjbpm/gpd/, 2008.

[34] The Eclipse Foundation: *Eclipse*. http://www.eclipse.org/, 2008.

[35] Workflow Patterns Initiative: *Workflow Patterns*. http://www.workflowpatterns.com/, 2008.

[36] Enterprise Integration Patterns: *Claim Check*. http://www.eaipatterns.com/StoreInLibrary.html, 2008.

[37] Sun Microsystems, Inc.: *Java Naming and Directory Interface (JNDI)*. http://java.sun.com/products/jndi/, 2008.

[38] Red Hat Middleware: *JBossESB Programmers Guide*. http://www.jboss.org/jbossesb/docs/4.4.GA/manuals/html/ProgrammersGuide.html, 2008.

[39] Marek Procházka: *Transakce*, Matematicko-fyzikální fakulta. http://dsrg.mff.cuni.cz/~prochazk/lectures/Tp000529.pdf, 2000.

[40] Wikipedia: *ACID*. http://en.wikipedia.org/wiki/ACID, 2008.

[41] Wikipedia: *Two-phase commit protocol*. http://en.wikipedia.org/wiki/Two-

phase_commit_protocol, 2008.

[42] Wikipedia: *Long-lived transaction*. http://en.wikipedia.org/wiki/Long-lived_transaction, 2008.

[43] Wikipedia: *Compensating transaction*. http://en.wikipedia.org/wiki/Compensating_transaction, 2008.

[44] IBM: *Web Services Transactions specifications*. http://www.ibm.com/developerworks/library/ws-coor/, 2008.

[45] Sun Microsystems, Inc.: *Java Transaction API (JTA), Version 1.1*. http://java.sun.com/javaee/technologies/jta/index.jsp, 2002.

[46] Wikipedia: *Common Object Request Broker Architecture*. http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture, 2008.

[47] Object Management Group, Inc.: *Object Transaction Service 1.2.1 Specification*. http://www.omg.org/cgi-bin/doc?formal/2001-11-03, 2003.

[48] Wikipedia: *IIOP*. http://en.wikipedia.org/wiki/IIOP, 2008.

[49] Red Hat Middleware: *JBossTransactionsVersionGuide*. http://www.jboss.org/community/docs/DOC-10804, 2008.

[50] Red Hat Middleware: *Getting Started With JBoss ESB*. http://www.jboss.org/jbossesb/docs/4.3.GA/manuals/html/GettingStarted.html, 2008.

[51] Ian Robinson's Weblog: *Is WS-Transaction useable in the real world today?* http://ianrobinson.blogspot.com/2007/08/is-ws-transaction-useable-in-real-world.html, 2007.

[52] Wikipedia: *WS-Coordination*. http://en.wikipedia.org/wiki/WS-Coordination, 2008.

[53] OASIS: *Web Service Atomic Transaction*. http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html, 2007.

[54] OASIS: *Web Service Business Activity*. http://docs.oasis-open.org/ws-tx/wsba/2006/06 , 2007.

[55] IBM: *Enabling optimistic locking*. http://publib.boulder.ibm.com/infocenter/radhelp/v6r0m1/index.jsp?topic=/com.ibm.etools.j2ee.ui.ws.ext.doc/topics/teoptloc.html, 2008.

[56] Microsoft: *Optimistic Concurrency Control*. http://msdn.microsoft.com/en-us/library/bb190073.aspx, 2008.

[57] Wikipedia: *Nested transaction*. http://en.wikipedia.org/wiki/Nested_transaction, 2008.

[58] Red Hat Middleware: *JBoss Transactions Forum - Distributed Transactions: What to do when coordinator crashes*. http://www.jboss.com/index.html?module=bb&op=viewtopic&t=134357&postdays=0&postorder=asc&start=0, 2008.

[59] Enterprise Integration Patterns: *Splitter*.

http://www.eaipatterns.com/Sequencer.html, 2008.

[60] Enterprise Integration Patterns: *Aggregator*. http://www.eaipatterns.com/Aggregator.html, 2008.

[61] Wikipedia: *Oracle RAC*. http://en.wikipedia.org/wiki/Oracle_RAC, 2008.

[62] Red Hat Middleware: *JBoss jBPM Forum - Job executor explained in a nutshell*. http://www.jboss.com/index.html?module=bb&op=viewtopic&t=121911&start=10, 2008.

[63] Wikipedia: *Swapping*. http://en.wikipedia.org/wiki/Paging, 2008.

[64] Red Hat Middleware: *Hibernate*. http://www.hibernate.org/, 2008.

[65] Wikipedia: *Granularity*. http://en.wikipedia.org/wiki/Granularity, 2008.

[66] ss64.com: *Oracle Datatypes*. http://www.ss64.com/orasyntax/datatypes.html, 2008.

[67] Sun Microsystems, Inc.: *jGuru: JDBC 2.0 Fundamentals*. http://java.sun.com/developer/onlineTraining/Database/JDBC20Intro/JDBC20.html, 2008.

[68] Red Hat Middleware: *Batch processing*. http://www.hibernate.org/hib_docs/reference/en/html/batch.html, 2008.

[69] Javaranch: *Batching Select Statements in JDBC*. http://www.javaranch.com/journal/200510/batching.html, 2005.

[70] Red Hat Middleware: *The Second Level Cache*. http://www.hibernate.org/hib_docs/reference/en/html/performance-cache.html, 2008.

[71] Red Hat Middleware: *JBoss Application Server*. http://www.jboss.org/jbossas/, 2008.

[72] Oracle Corporation: *The Official Oracle Site*. http://www.oracle.com/, 2008.

[73] Sun Microsystems, Inc.: *Java SE*. http://java.sun.com/javase/, 2008.

[74] Sun Microsystems, Inc.: *Java EE*. http://java.sun.com/javaee/, 2008.

[75] Red Hat Middleware: *JBossMQ*. http://www.jboss.org/community/docs/DOC-10525, 2008.

[76] Red Hat Middleware: *JBoss Transactions*. http://www.jboss.org/jbosstm/, 2008.

[77] Refnes Data: *XML Technologies*. http://www.w3schools.com/xml/xml_technologies.asp, 2008.

[78] Apache Software Foundation: *Apache Logging Services Project*. http://logging.apache.org/, 2008.

[79] W3C: *Web Services Activity*. http://www.w3.org/2002/ws/, 2008.

[80] Red Hat Middleware: *JBoss Tools Project*. http://www.jboss.org/tools/, 2008.

[81] Apache Software Foundation: *Apache Ant*. http://ant.apache.org/, 2008.

[82] Novell, Inc.: *Linux Server: SUSE Linux Enterprise Server by Novell.* http://www.novell.com/products/server/, 2008.

[83] Red Hat, Inc.: *Fedora project*. http://fedoraproject.org/, 2008.

[84] Microsoft: *Windows Vista*. http://www.microsoft.com/, 2008.

[85] Red Hat Middleware: *JBoss Messaging Features*. http://www.jboss.org/file-access/default/members/jbossmessaging/freezone/docs/userguide-1.4.0.SP3/html_single/index.html#features, 2008.

[86] Red Hat Middleware: *JBoss Messaging*. http://www.jboss.org/jbossmessaging/, 2008.

18 Přílohy

Přiložené CD

Přiložené CD obsahuje následující data:

doc/	dokumentace • technologická dokumentace • dokumentace k business procesu Member fees
src/	zdrojové soubory
install/	ukázková instalace
diplomova_prace.pdf	tato diplomová práce v pdf formátu