Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Management distribuovaného výpočetního prostředí

Plzeň, 2004 František Dvořák

Zadání

1) Seznamte se s funkcemi systému pro management distribuovaného výpočetního prostředí (DVP) s ohledem na specifika DVP ZČU – Orion.
2) Proveďte analýzu potřeb managementu DVP s důrazem na stanovení hlavních funkcí a rozhraní vrstvy aplikační logiky ve vícevrstvém modelu tvorby informačního systému.
3) Navrhněte sadu funkcí aplikační logiky managementu DVP (její rozhraní vůči prezentační logice).
4) Implementujte vybranou část aplikační logiky a prokažte její funkcionalitu realizací jednoduché prezentační vrstvy.

Abstrakt (česky)

Tato diplomová práce se jmenuje "Management distribuovaného výpočetního prostředí". Funkce managementu na Západočeské univerzitě v Plzni je centrální administrace dat a spolupráce s ostatními databázemi: registry strojů a osob.

Hlavním cílem návrhu je aplikační logika managementu (rozhraní vůči prezentační logice) a ověření funkčnosti na implementaci části aplikační logiky. Programovacím jazykem je Java a implementace je na platformě J2EE – Java 2 Enterprise Edition.

Mají být ukázány možnosti třívrstvého návrhu tohoto systému: zapouzdřit spolupráci se všemi databázemi do prostřední aplikační vrstvy a umožnit budoucí změny rozhraní registrů bez ovlivnění koncové prezentační vrstvy.

Abstrakt (english)

This thesis is called „Distributed computing environment management“. The function of the management in University of West Bohemia is central data administration and cooperation with other databases: machine and person registers.

Primary goal of design is application logic of the management (the interface toward presentation logic). As program language is Java and the implementation is in J2EE platform – Java 2 Enterprise Edition.

Possibilities of three-layer design shall be presented: encaspulate the cooperation with all databases into middle application logic and make possible future changes of registry interface without interference in the front presentation layer.

Obsah

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

Obsah

V Plzni dne 11.8.2004 František Dvořák

1. Úvod

Předmětem této diplomové práce je management distribuovaného výpočetního prostředí na Západočeské univerzitě v Plzni. Cílem je navrhnout aplikační logiku správy managementu a ověřit funkčnost na implementaci části aplikační logiky.

Základní funkcí managementu distribuovaného výpočetního prostředí je centrální administrace dat. Jádrem této diplomové práce je zpracovat analýzu MDVP jako třívrstvé aplikace.

Vrstvy MDPV budou následující:

1) Datová základna – data uložená v relační databázi, přístup k nim je přes SQL.
2) Aplikační logika – sada vysokoúrovňových operací realizující funkcionalitu, která se od managementu DVP očekává.
3) Prezentační logika – pouze realizuje prezentaci dat uživateli a nástroje pro změny dat voláním funkcí 2. vrstvy.

   Může zahrnovat grafického klienta, WEB klienta, skripty či portlet pro portál.

Vrstva datové základny

Dělí se na dvě části:

- data specifická pro management DVP (login, kvóta, konfigurace stroje, atd.)
- data obecná, ze společných registrů (registr osob, strojů či místností)

V optimálním případě se zde spolupracuje s projektem registrů, a to tak, že v datové základně jsou pouze data managementu DVP a obecná data se získávají z registrů.

Implementace vazby mezi těmito částmi není zatím pevně dána. Může jít o např. o napojení tabulek: registry budou ve formě DB tabulek, jejichž obsah vidíme, ale stará se o ně jiná aplikace. Další možností je komunikace přes nějaké API (middleware), tj. volání funkcí, které poskytují informace a služby, které potřebujeme. Jedním z cílů aplikační logiky managementu je právě abstrakce od způsobu přístupu k registrům. Při případné změně rozhraní bude potřeba pozměnit pouze implementaci jádra managementu DVP.

Vrstva aplikační logiky

S první vrstvou komunikuje aplikační logika přes SQL a případně také přes API registrů. Poskytuje definované vysokoúrovňové operace (změnit kvótu, zaregistrovat uživatele, zablokovat uživatele, atd.). Měla by poskytovat takovou sadu funkcí, aby nemusela prezentační

logika přistupovat k nižším vrstvám.

Funkce rozhraní musí být k dispozici širokému spektru potencionálních implementací prezentační logiky, ideovým vzorem tohoto přístupu jsou web services. Postačí, když se bude aplikační logika skládat z tříd a metod v Javě a bude schopna běžet na nějakém aplikačním serveru.

Vrstva prezentační

S použitím rozhraní aplikační logiky (bez přístupu k databázi) implementuje vlastní aplikaci tak, jak ji vidí uživatel.

Typickým použitím bude např. WWW aplikace, nejlépe snadno převeditelná v nativní portálovou aplikaci (portlet).

Druhým použitím budou nástroje příkazové řádky (pro dávkové administrativní zásahy) a zejména skripty realizující vlastní řízení distribuovaného výpočetního prostředí (vytváření souborových systémů, změny kvót, publikace do LDAP apod.). Cílem by mělo být, aby i tyto skripty a příkazy používaly pouze API aplikační logiky a nikoliv databázi.

V rámci diplomové práce bude prezentační logika pouze ukázková a jako nástroj na testování a demonstraci funkce aplikační logiky.

Tato první úvodní kapitola byla seznámení s řešenou problematikou. Byla zde popsána struktura managementu distribuovaného výpočetního prostředí a očekávané funkce.

Další, druhá kapitola, je teoretická část, která pojednává o architekturách softwarových systémů a o platformě J2EE, která byla zadána pro realizaci diplomové práce.

Třetí a čtvrtá kapitola je praktická část diplomové práce. Třetí, stěžejní kapitola, obsahuje návrh aplikační logiky, čtvrtá kapitola popisuje návrh a realizaci ověřovací implementace.

2. Vícevrstvý model tvorby informačního systému

Podívejme se nejprve blíže na vícevrstvé architektury programových systémů.

2.1. Aplikační vrstvy

Vzory architektury lze rozdělit na různé druhy podle toho, kolik definují aplikačních vrstev. Výraz “vrstvy” se odkazuje na různé části aplikace: prezentační, definici logiky a defi-

nici dat.

Prezentační vrstva dodává uživatelské rozhraní a řídí pohyb uživatele v aplikaci. Požadavky na tuto vrstvu plynou z lidských faktorů jako je vnímání, reakční doby a srozumitelnost.

Aplikační vrstva modeluje logiku daného problému. Například bankovní aplikace může mít interní reprezentaci kont zákazníků a práci s konty modelovat pomocí logiky pro realizaci vkladů a výběrů.

Datová vrstva modeluje požadavky na uložení informací. Aplikace musí tyto požadavky převést na požadavky na použitý systém pro uložení dat. Tím může být například systém souborů nebo relační databáze.

Na základě těchto tří vrstev definujeme tři základní typy aplikací:

- *Jednovrstvé* aplikace jsou monolitické a neoddělují prezentační logiku, aplikační logiku a logiku pro práci s daty.
- *Dvouvrstvé* aplikace oddělují prezentační a aplikační logiku od logiky pro práci s daty.
- *Třívrstvé* aplikace oddělují prezentační, aplikační logiku a logiku pro práci s daty do oddělených komponent.

Těmito třemi uvedenými modely nelze samozřejmě postihnout všechny architektury. Existují i komplikovanější návrhy.

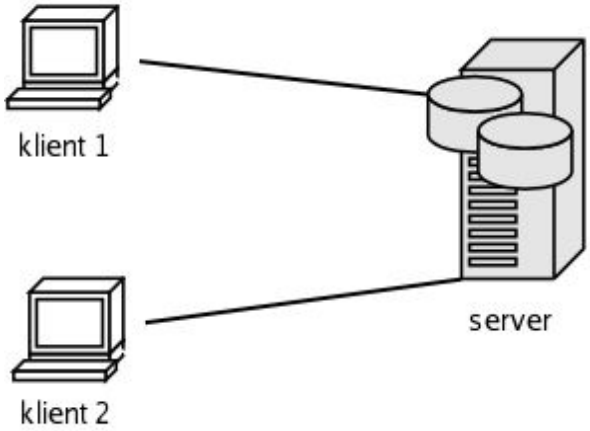
2.1.1. Jednovrstvé architektury

Jednovrstvé architektury se nesnaží od sebe navzájem oddělit různé části aplikace. Moderní metody vývoje software degradovaly takovýto způsob tvorby aplikací na způsob vhodný pouze pro tvorbu malých utilit spouštěných z příkazového řádku a pro neinteraktivní dávkové zpracování. Toto bylo způsobeno především příchodem moderních systémů typu klient-server, což je druh dvouvrstvé architektury.

2.1.2. Dvouvrstvé architektury

Dvouvrstvé architektury oddělují prezentační a aplikační logiku od logiky pro přístup k datům. Databázové aplikace, jednoduché WWW aplikace a všechny aplikace typu klient-server demonstrují použití tohoto typu architektury. Mnohé moderní interaktivní aplikace také

používají tuto architekturu především pro zajištění sdílení dat.



Obrázek 1: dvouvrstvá architektura

Dvouvrstvé architektury představují lepší alternativu k jednovrstvým architekturám, pokud jsou důležité požadavky na sdílení dat a na jejich centralizovanou správu. Jejich výhoda také bývá v tom, že většinou je snadnější jim porozumět a implementovat je. Avšak mají také své nevýhody.

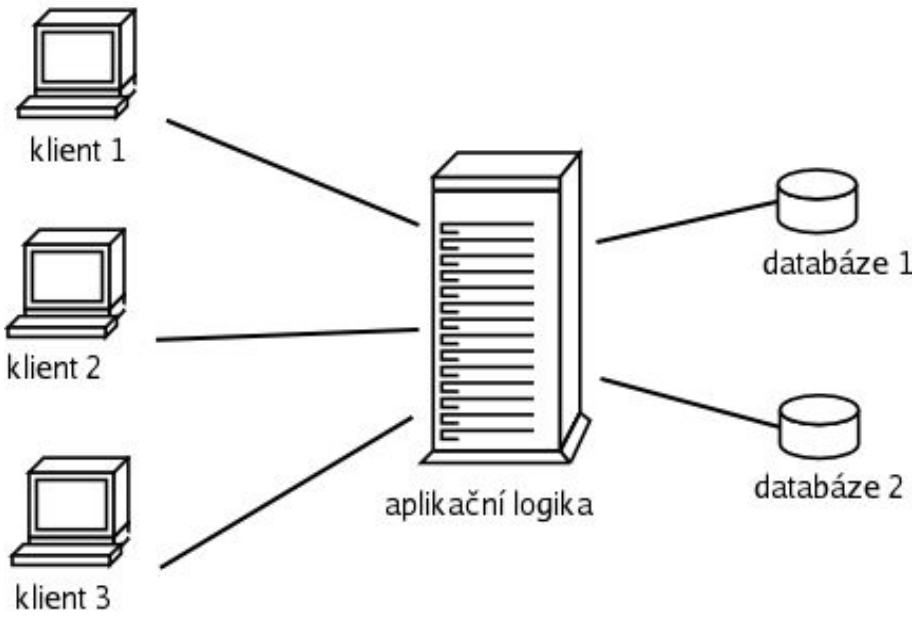
Dvouvrstvé architektury staví na myšlence, že aplikace by měly zpracovávat data lokálně, ale ukládat by je měly ve sdíleném centrálním serveru. To je ale příliš zjednodušující. V případě firemní databázové aplikace musí každá běžící instalace aplikace samostatně zpracovávat sdílená data v podstatě stejným způsobem. Pokud aplikace zpracovávají data lokálně, každá z nich musí duplikovat případnou společnou logiku. Pro odstranění této duplicity lze využít uložených procedur, které moderní databáze podporují. Tím se společná aplikační logika přesune do databázového serveru. Toto řešení je však problematické tím, že narušuje jasné oddělení aplikační vrstvy logiky a logiky pro přístup k datům. Některé databáze jako Oracle nabízí možnost volání kódu Javy přímo v prostředí databázového serveru. Avšak často platí, že čím více aplikační logiky je implementováno přímo v databázovém serveru, tím více se znepřehlední struktura aplikace a její vazby na další aplikace. Tento postup také váže aplikaci na konkrétní databázový server, protože tvorba uložených procedur není mezi databázovými servery různých výrobců standardizována.

Dále také musí moderní aplikace často přistupovat k více databázím. Dvouvrstvé aplikace

musí mít logiku pro práci s více databázemi v aplikační vrstvě. Tím se přesunuje logika pro přístup k datům z databázového serveru do aplikace. Aplikační programátor musí potom dobře vědět, které databáze obsahují jaká data a také musí velmi dobře znát datový model každé databáze.

2.1.3. Třívrstvé architektury

Třívrstvé architektury oddělují prezentační logiku, aplikační logiku a logiku pro přístup k datům do tří různých částí. Typickou třívrstvou architekturu ukazuje následující obrázek:



Obrázek 2: třívrstvá architektura

Hlavní výhodou třívrstvé architektury je podpora jasnějšího oddělení a zapouzdření funkcí aplikace. Prezentační logika realizuje požadavky na uživatelské rozhraní a logika pro přístup k datům zahrnuje realizaci struktury fyzického datového modelu a využití možností databázového serveru. Prostřední vrstva je aplikační vrstvou, která se také zabývá modelováním sémantiky logiky daného problému. Výhodou společné aplikační vrstvy pro všechny klienty je abstrakce od způsobu komunikace s databází. Je možné mít v systému několik databází bez toho, aby o tom musela vědět každá klientská aplikace.

Dvouvrstvá architektura umožnila odstranit z klientské části aplikace datovou vrstvu, tří-

vrstvá architektura k tomu dále umožňuje přidělit různým částem aplikace vlastní hardwarové prostředky. Možnost přidání hardwarových prostředků jednotlivým vrstvám výrazně zlepší škálovatelnost aplikace.

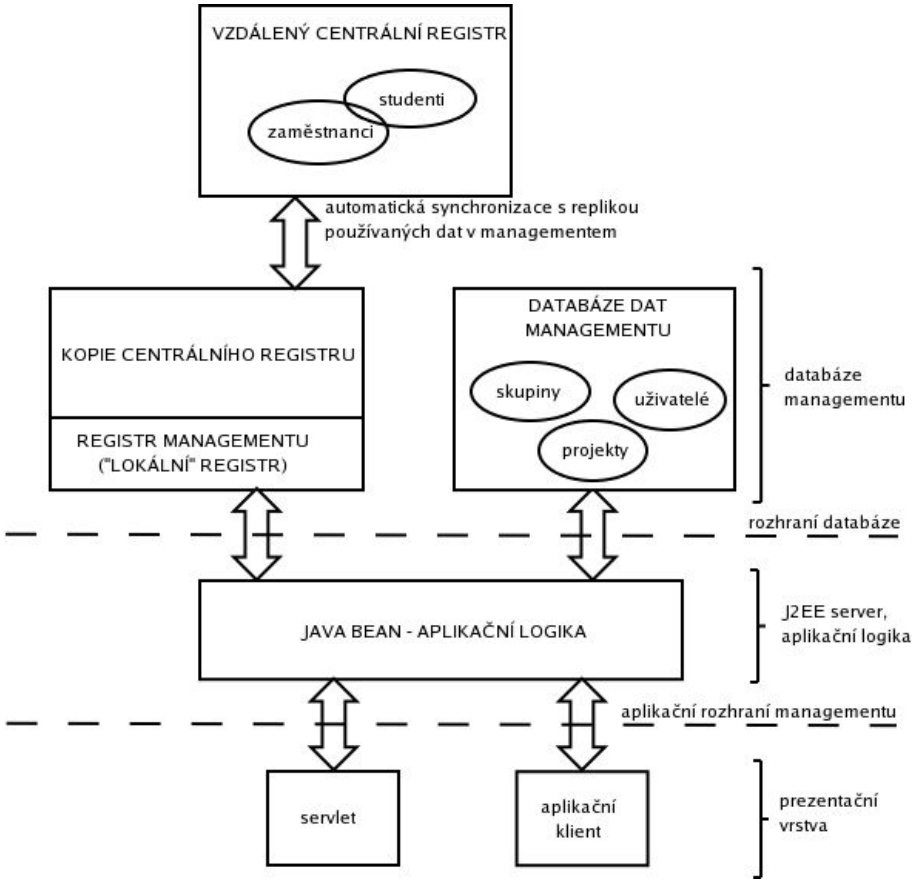
Třívrstvá architektura, ačkoliv je lépe škálovatelná a flexibilnější, má i své nevýhody. Vyžaduje přídavnou síťovou vrstvu, která snižuje výkon a komplikuje údržbu aplikace a řešení problémů. Také implementovat třívrstvou architekturu je složitější. Navíc, síť není nikdy zcela transparentní. Otázky jako zpoždění a spolehlivost hrají u třívrstvé architektury větší roli.

Výsledkem je, že mnohé aplikace lze realizovat mnohem jednodušeji jako jedno či dvouvrstvé aplikace.

2.2. Architektura managementu distribuovaného výpočetního prostředí

Na následujícím obrázku jsou vyznačeny tři vrstvy navrhovaného managementu distribuovaného výpočetního prostředí:

2. Vícevrstvý model tvorby informačního systému



Obrázek 3: architektura managementu distribuovaného výpočetního prostředí

- Horní vrstva je datová vrstva uložená v databázi managementu. Některá data se získávají z odděleného centrálního registru. Rozhraní datové vrstvy zůstane skryto uvnitř managementu.
- Prostřední vrstva je aplikační logika. Jádrem diplomové práce je hlavně její poskytované rozhraní pro prezentační vrstvu.
- Dolní vrstva představuje prezentační vrstvu.

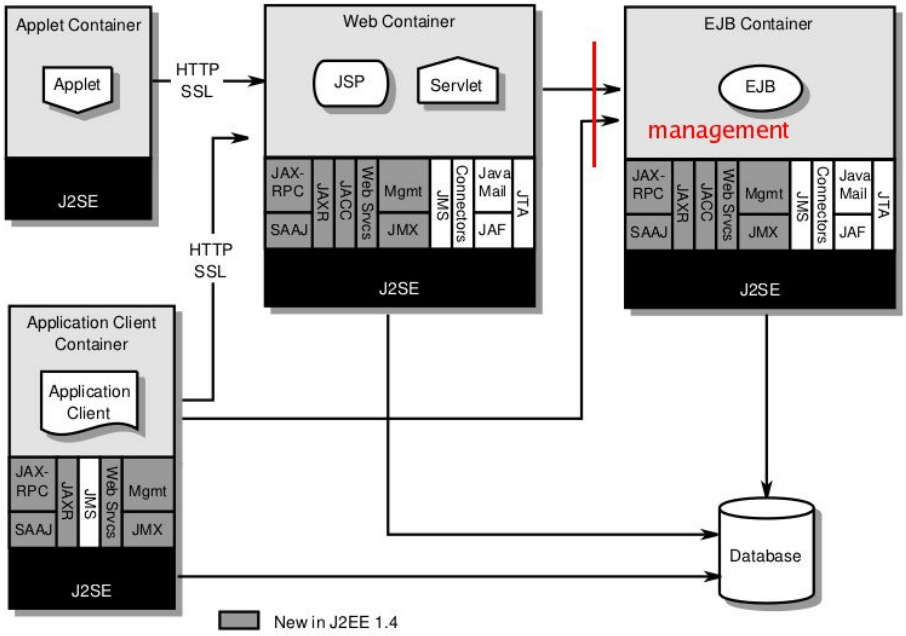
2.3. Architektura J2EE

Platforma Java 2 Enterprise Edition je navržena pro distribuované vícevrstvé programové systémy. Základem platformy J2EE jsou komponenty. Každá komponenta má určitý účel a

smí, ale nemusí, běžet na odděleném serveru.

Důležitou součástí platformy J2EE jsou *Enterprise Java Beans* (EJB). *Enterprise Java Bean* je komponenta, která běží na straně serveru. Klienty EJB mohou být servlety, applety i jiné aplikace, které dokonce nemusí být psány v jazyce Java. Klientem může být i další EJB, což rozšiřuje možnosti použití EJB v distribuovaném vícevrstvém prostředí.

Následující obrázek ukazuje základní architekturu J2EE. Šipky reprezentují směr přístupu mezi jednotlivými částmi J2EE:



Obrázek 4: architektura platformy J2EE.

Application Client Container poskytuje prostředí pro aplikační klienty, *Web Container* pro JSP stránky a servlety a *EJB Container* pro Enterprise Java Beans. Aplikační klienti, JSP stránky, servlety i EJB smí přistupovat do databází přes rozhraní JDBC. V této diplomové práci bude jedinou komponentou přistupující k databázi EJB.

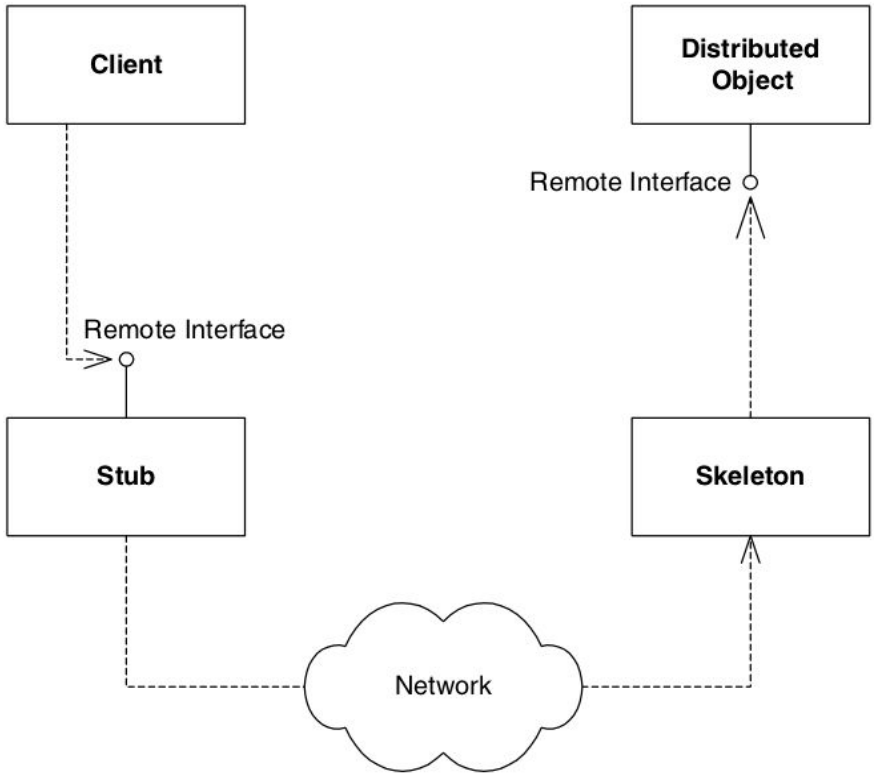
Jádro managementu bude v EJB Containeru. Předmětem diplomové práce bude aplikační rozhraní červeně vyznačené na obrázku. Prezentační vrstva – klienty – mohou být např. v Application Client Containeru nebo ve Web Containeru.

2.3.1. Enterprise Java Beans

Jsou tři základní typy Enterprise Java Beans:

- *Session EJB* – modelují proces. Proces může být aplikační logika, přístup k databázi a k ostatním beanům.
- *Entity EJB* – modelují data. K dispozici jsou javovské objekty, které reprezentují informace z databáze. Jsou definovány vztahy mezi objekty a je udržována konzistence s databází.
- *Message-Driven EJB* – souvisejí s procesem. Jsou volány zasíláním asynchronních zpráv. Tyto EJB rozšiřují použitelnost J2EE na další specifické typy aplikací.

Každý typ EJB je distribuovaný objekt, který může být volán z lokálního i vzdáleného systému. Způsob volání je vidět z následujícího obrázku:



Obrázek 5: vzdálené volání metod

Na straně klienta je umístěna tzv. *spojka* (stub). *Spojka* je zodpovědná za síťovou komu-

nikaci u klienta. Stará se o vlastní komunikaci v síti, konverzi parametrů a výsledků. Vše zapouzdřuje tak, aby byla práce se sítí klientovi skryta – tzv. *lokální transparentnost*.

Zprávy spojky přijímá *skelet* (kostra, skeleton) na serveru. Ten skrývá síťovou komunikaci před EJB. *Skelet* ví, jak reprezentovat data přijatá ze sítě, a přijímané parametry konvertuje do reprezentace v Javě. Skelet potom předá řízení distribuovanému objektu a distribuovaný objekt vrací výsledky, které skelet předá zpět spojce na klientovi.

Distribuovaný objekt a spojka na klientovi implementují stejné rozhraní – *remote interface*.

2.3.2. Remote Method Invocation

Komunikace v distribuovaném prostředí J2EE je založena zejména na vzdáleném vykonávání procedur (*Remote Method Invocation*). Při implementaci RMI v Javě musely být řešeny různé problémy – např. způsob přenosu Javovských objektů a referencí po síti a reakce na chyby, ať při přenosu nebo při neočekávaném ukončení aplikace.

Podívejme se na předávání argumentů. Existují dva způsoby: předávání hodnotou a předávání odkazem.

- Předávání hodnotou – hodnota nebo objekt se konvertuje do a z „bitového tvaru“. Řešení v Javě je velmi elegantní. Každý objekt, který se bude předávat, musí mít schopnost serializace. Prakticky se jedná o implementaci prázdného rozhraní `Serializable`, u složitějších objektů je nutné řešit i další problémy. Takovéto objekty lze pak nejen přenášet po síti, ale lze je také např. zapisovat do souboru.
- Předávání odkazem – klient posílající referenci odešle spojku (stub), která také implementuje rozhraní `Serializable`. Ta poskytne serveru callback, pomocí kterého může server měnit vzdálený objekt.

2.3.3. Java Naming and Directory services

JNDI – jmenné a adresářové služby – je aplikační rozhraní J2EE, které umožňuje specifikovat v síti uživatele, stroje, objekty a služby. Služeb JNDI využívají rozhraní EJB, RMI-IIOP, JDBC a další.

JNDI zahrnuje dva typy služeb:

a) jmenné služby – asociuje jména s objekty. Tzv. *binding* (vazba).

b) adresářové služby – jmenné služby rozšířené o adresáře. Adresář je systém adresářových objektů, které jsou propojeny do hierarchické stromové struktury. Struktura připomíná databázi, a občas to bývá i tak implementováno.

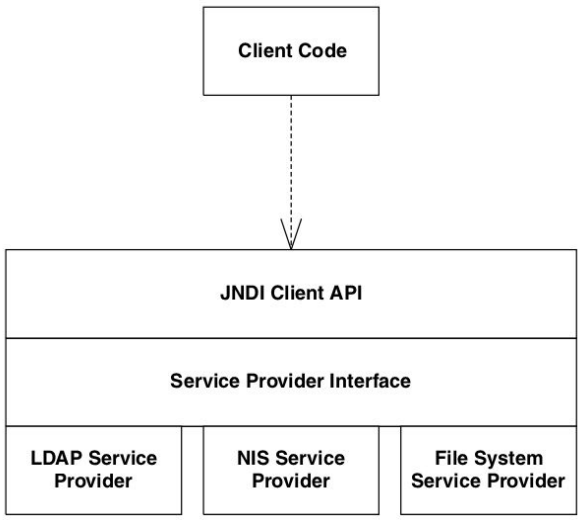
JNDI bylo navrženo jako „most“ nad existujícími jmennými a adresářovými službami. Umožňuje jednotný přístup aplikací ke všem podporovaným jmenným službám: LDAP, NIS, Novel NDS, SLP, CORBA, RMI-IIOP, atd.

Architektura JNDI má dvě části:

a) klientské aplikační rozhraní

b) Service Provider Interface (SPI)

Klientské aplikační rozhraní dovoluje provádět adresářové operace a SPI je rozhraní pro realizaci jednotlivých druhů jmenných služeb v JNDI:



Obrázek 6: architektura JNDI – jmenných a adresářových služeb

Máme-li vzdáleného klienta komunikujícího se serverem, pak je jednou z nejdůležitějších informací protokol a vzdálený server. Tyto informace se předají JVM buď při spouštění klienta, nebo se nakonfiguruje Client Container, pod kterým klient poběží.

3. Návrh rozhraní aplikační logiky managementu DVP

Hlavním cílem této diplomové práce je navrhnout aplikační rozhraní managementu dis-

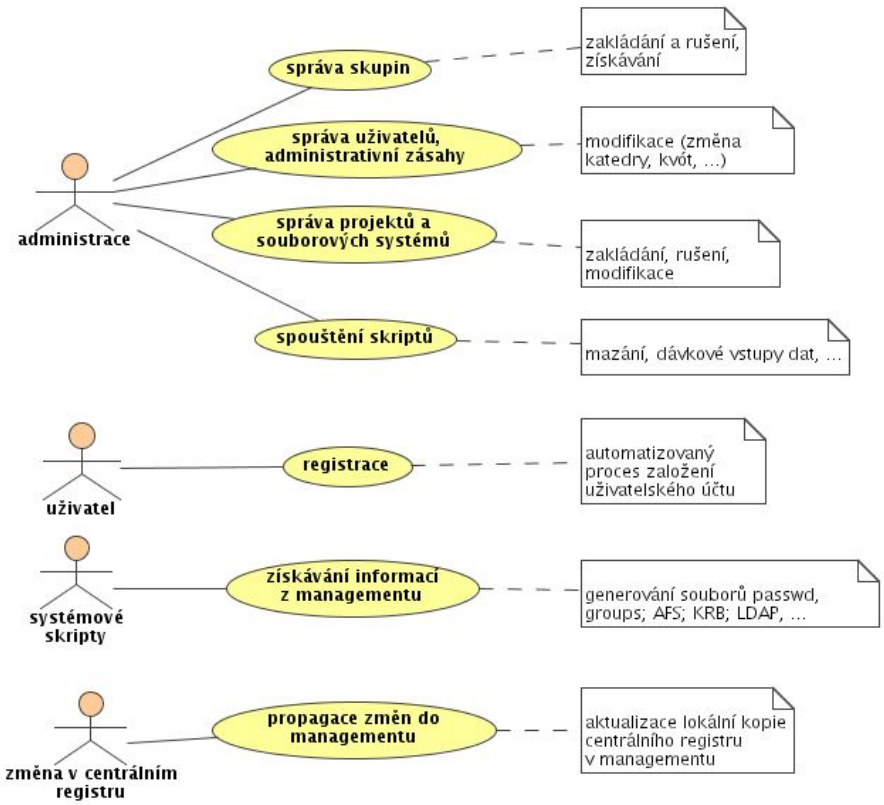
tribuovaného výpočetního systému. Prvotní představa o funkci systému a kladených požadavcích bude ukázána v následující kapitole.

3.1. Analýza potřeb managementu distribuovaného výpočetního prostředí

Management distribuovaného výpočetního prostředí je komplexní úloha. V managementu se evidují skupiny, uživatelé, stroje, skupiny strojů (clustery), sítě, souborové systémy, atd. a podstatná část informací je k dispozici odděleně v centrálních registrech.

Aby nebyla úloha tak rozsáhlá, vyškrtneme z úlohy jednu část, která je relativně nezávislá – správa strojů a clusterů. Z hlediska návrhu softwarových projektů to sice není správný postup, ale vycházíme z už existujícího managementu, a podle praktických zkušeností s managementem si to doufejme můžeme dovolit.

Definujeme si tedy cíl návrhu pomocí diagramu užití:



Obrázek 7: diagram užití

3. Návrh rozhraní aplikační logiky managementu DVP

U typických návrhů softwarových projektů bývá diagram užití podrobnější, protože popisuje celou aplikaci jako celek, často včetně „podpůrných“ use-casů pro návrh grafického rozhraní. Softwarový návrh v této diplomové práci je zaměřen na aplikační rozhraní managementu. To bude stejné pro každou koncovou aplikaci a diagram užití zde proto musí být obecnější a platný pokud možno pro co nejširší okruh budoucích aplikací.

3.1.1. Administrace

a) práce se skupinami managementu

   V managementu výpočetního distribuovaného systému se sdružují uživatelé do skupin. Každá skupina může zároveň obsahovat i další podskupiny a jeden uživatel může být ve více skupinách. Výsledkem je stromová struktura skupin a uživatelů.

   Očekáváme možnost získání seznamu skupin a uživatelů, a to jak rekurzivně, tak nerekurzivně. Dále chceme umožnit vytváření a rušení skupin, modifikaci skupin a přidávání nebo odebírání uživatelů a podskupin do skupiny a ze skupiny.

b) práce s uživateli managementu, administrativní zásahy

   Chceme, aby existovaly služby pro administrátorské rozhraní, které by umožňovaly měnit některé údaje v managementu – nastavovat kvóty uživatelům, blokovat uživatele, měnit další údaje.

   Bude nutné uvažovat také speciální uživatelské účty bez vazby na centrální registr osob informačního systému.

c) souborové systémy (projekty, domácí adresáře, ...)

   Budeme potřebovat zakládat, rušit a spravovat projekty a souborové systémy. Každý projekt nebo souborový systém má jednoho vlastníka, a každý uživatel může vlastnit více projektů a souborových systémů.

d) jednorázové skripty (mazání, dávkové vstupy dat)

   Zde chceme možnost např. mazání většího množství dat nebo můžeme chtít dávkové vstupy dat. Z hlediska návrhu aplikačního rozhraní toto použití zřejmě nebude podstatné, protože pro skripty nebudou potřeba speciální služby managementu. Tento případ je však důležitý z hlediska implementace. Některé z klientských aplikací využívajících služeb navrhovaného rozhraní budou ovládány z příkazové řádky.

3.1.2. Registrace uživatelů

Registrace je automatizovaný proces získání uživatelského konta. Pro pochopení toho, co přesně má MDVP při registraci dělat, je nutné se blíže podívat na pojem *uživatel*.

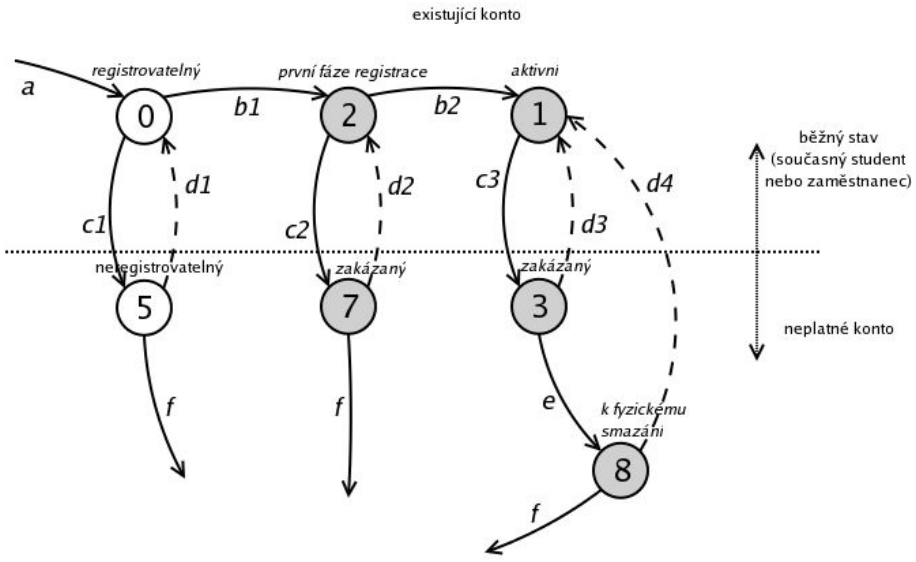
Z hlediska managementu je entita *uživatel* rozdělena na dvě části: *osoba* a *uživatelské konto*:

- *Osoba* odpovídá skutečné osobě. Ta bude evidována buď v centrálním registru osob informačního systému nebo lokálně v databázi managementu distribuovaného výpočetního prostředí.
- *Uživatelské konto* bude evidováno v databázi managementu.

Jedna osoba může mít obecně několik uživatelských kont, tj. vazba mezi osobou a uživatelem je 1:M. Ale to jsou jen speciální případy, např. u administrátorských kont. Po registraci uživatelských kont přes aplikační rozhraní managementu proto chceme pouze vazbu 1:1.

3.1.2.1. Životní cyklus uživatelského účtu

Každý uživatel má v managementu distribuovaného výpočetního systému svůj životní cyklus, může se nacházet v různých stavech:



Obrázek 8: stavy uživatele v managementu

Vybarvené stavy ve stavovém diagramu odpovídají osobě s vytvořeným uživatelským

kontem, nevybarvené stavy značí osobu bez konta.

V horní části diagramu uživatel není zakázán, ale aby mohl v systému pracovat, potřebuje být ve stavu 1. Dolní část diagramu odpovídá situaci, kdy osoba nesmí konto používat nebo na něj nemá nárok – je zablokováno.

Ve způsobu registrace se vychází ze současného řešení – projektu Moira. Registrace uživatele probíhá ve dvou fázích. První fází je rezervování přihlašovacího jména, druhou fází je přidělení zdrojů a dokončení registrace: nastavení hesla a vytvoření domovského adresáře (souborového systému). Důvodem dvou fází je rozdělení jedné komplexnější operace, která by musela být atomická, na dvě části.

Číslování stavů bylo převzato z projektu Moira. Vysvětlení jednotlivých operací:

a) vložení informace o osobě do managementu

   Je nutné mít dvě kategorie osob:

   - centrálního registru osob – vazba na primární aplikaci garantující způsob pořízení záznamu
   - lokální registr v databázi managementu – bez pevné vazby na ZČU

b) registrace uživatele

   Jedná se o proces získání uživatelského konta. Registrace má dvě fáze, přičemž uživatel může zůstat v mezistavu.

c) blokace uživatele

   Na základě informace, že osoba ztrácí nárok na konto, se konto zablokuje. Zablokovat konto lze v různých stavech.

d) odblokování uživatele

   Na základě informace, že osoba získává zpět nárok na konto, se konto odblokuje. Opět se vychází z různých stavů. Pokud již konto bylo odstraněno z managementu, jedná se proces v a).

e) ochranná lhůta

   Blokovaný uživatel nesmí být po určitou dobu smazán.

f) fyzické smazání uživatele

   Jde o uvolnění zdrojů – vymazání informací z databáze, smazání domovského adresáře,

pokud byl vytvořen, atd.

Smazat lze pouze konto, které je zakázané. Bylo-li mazané konto funkční (tj. ve stavu 1), je zařazen ještě mezistav – čekací doba před fyzickým smazáním dat.

3.1.3. Systémové skripty

Hlavními službami v této kategorii je zpřístupnění dat managementu systémovým skriptům. Hlavní účel skriptů bude generování konfiguračních souborů, např.:

- passwd, groups
- AFS
- KRB
- LDAP
- a další

3.1.4. Reakce na změny v registru

Aplikační rozhraní by mělo poskytovat funkce pro synchronizaci lokální kopie centrálního registru v databázi managementu s centrálním registrem. Dále budou potřeba funkce na evidenci osob v lokálním registru managementu.

3.2. Návrh aplikačního rozhraní

Elegantní by byl objektový způsob – objekty managementu a jim odpovídající metody, např. objekt skupiny s metodami na přidávání a odebírání členů skupiny. Takováto realizace je možná, ale každý objekt (uživatel, skupina nebo souborový systém) by potřeboval zpětný odkaz na java bean.

Proto je tedy v rámci designu systému jako základ aplikačního rozhraní zvolen pouze javovský bean a její výkonné funkce. Ty pak pracují s objekty managementu a jediný účel objektů je uchovávání a předávání dat – atributů.

Výkonné funkce lze rozdělit do těchto kategorií:

a) vyhledávání a získávání informací: Bude třeba vyhledávat informace podle určitých kritérií. Kritéria by mohla být teoreticky jakákoliv. I při pokusu o návrh s co

nejobecnějšími kritérii se může objevit nutnost přidání dalších, složitějších kritérií. Prakticky se proto bude vyhledávat pouze podle atributů: podle regulárních výrazů v případě řetězců a podle rovnosti v případě jiných typů atributů. Jestliže bude potřeba vyhledávat informace na serveru podle jiných složitějších kritérií, aplikační rozhraní se rozšíří o příslušné specializované vyhledávací funkce.

b) přidávání dat – bude se jednat buď o předávání dynamicky vytvářených objektů aplikačnímu rozhraní nebo o volání specializovaných funkcí
c) modifikace dat – data budou modifikována podle změněných atributů objektu
d) mazání dat

3.2.1. Reakce na chybové stavy nebo neúspěšná volání

Existují dvě možnosti, jak reagovat na kritické chyby anebo neúspěšná volání funkcí:

- informování o úspěšnosti nebo neúspěšnosti (návratová hodnota funkce, nastavení příznaku, atd.)
- vyvolání výjimky

Obecným pravidlem je reagovat na chyby vždy vyvoláním výjimky, ale otázkou zůstávají běžná provozní selhání, pomocí nichž je pak dále řízen tok programu (např. neúspěšná rezervace přihlašovacího jména). Tam by použití výjimek nemuselo být vhodné.

Protože je nutné jakékoliv neúspěšné volání stejně vždy ošetřit, je vlastně i mechanizmus výjimek v jazyka Java velmi vhodný. Nakonec byl tedy zvolen pro indikaci jak chybových stavů tak i „běžných“ neúspěšných volání mechanismus výjimek.

Pro výjimky byla zavedena zvláštní třída `ManagementException`.

Dále je zde otázka, jak konkrétně implementovat různé typy výjimek a chyb? Často to bývá implementováno pomocí dědičnosti objektů. Z důvodu jednoduchosti se omezíme se pouze na jednu výjimku a typ chyby se bude rozlišovat pomocí atributu v objektu výjimky.

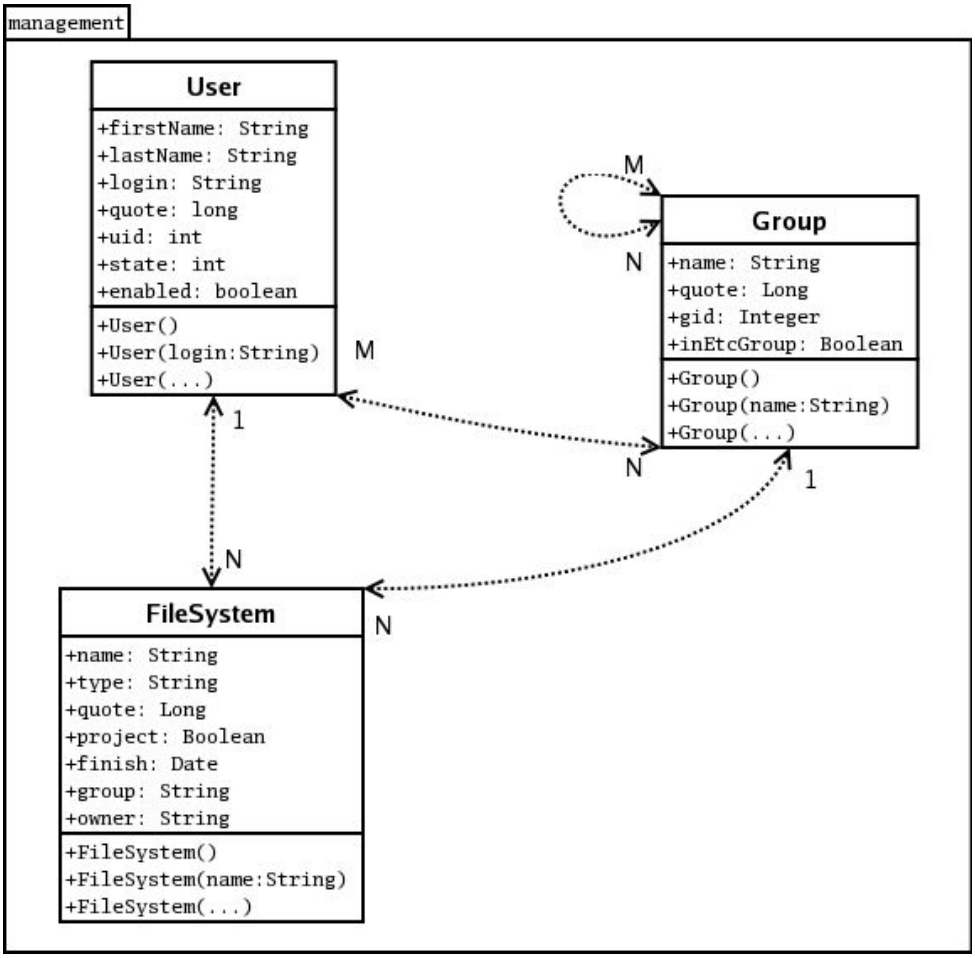
3.2.2. Objekty managementu

V navrhované části managementu DVP jsou tři základní typy objektů, které přibližně odpovídají entitám v datovém návrhu:

- `User` – reprezentace uživatele
- `Group` – reprezentace skupiny uživatelů

- `FileSystem` – reprezentace souborového systému

Následující obrázek ukazuje logické vazby objektů aplikačního rozhraní managementu. Jedná se v podstatě o běžný class diagram, ale uvedené vazby mezi objekty nebudou realizovány přímo referencemi na objekty, ale budou realizovány pouze uvnitř managementu (tj. v datové nebo aplikační vrstvě):



Obrázek 9: objekty aplikačního rozhraní managementu s vyznačenými virtuálními vazbami

3.2.2.1. Objekt `User`

Abstraktní pohled na uživatele v managementu. Z hlediska managementu je sice entita „uživatel“ rozdělena na dvě části, *osoba* a *uživatelské konto*, ale to zůstane v aplikačním rozhraní skryto.

Objekt reprezentující uživatele managementu bude mít následující atributy:

- `String firstName` – křestní jméno uživatele
- `String lastName` – příjmení
- `String login` – přihlašovací jméno
- `int uid` – UNIXovské id uživatele
- `String primaryGroup` – název primární skupiny, kam uživatel patří
- `int state` – stav uživatele (použití tohoto atributu by mělo být pouze informační – ke čtení). Tento atribut odpovídá čísle stavu uživatelského konta.
- `boolean enabled` – stav uživatelského konta, zda je povoleno či zakázáno

Konstruktory objektu `User`:

a) `User()` – vytvoření prázdného objektu `User` (bez uvedených atributů)

b) `User(String login)` – vytvoření objektu uživatele pouze s uvedeným přihlašovacím jménem (použití takového konstruktoru bude např. pro objekt jako vyhledávací kritérium)

c) `User(String firstName, String lastName, String login, String group, Long quote, Integer uid, Integer state)` – vytvoření objektu reprezentujícího uživatele

3.2.2.2. Objekt `Group`

Objekt `Group` reprezentuje v managementu množinu uživatelů a skupin.

Atributy:

- `String name` – název skupiny
- `Long quote` – standardní kvóta uživatelů ve skupině
- `Integer gid` – UNIXovské group ID
- `Boolean inEtcGroup` – zda je skupina uvedena v souboru `/etc/group`

Konstruktory objektu `Group`:

a) `Group()` – vytvoření objektu reprezentující skupinu bez uvedených atributů.

b) `Group(String name)` – vytvoření objektu skupiny pouze s uvedením jména skupiny, ostatní atributy budou prázdné (použití tohoto konstruktoru je např. pro ojekt jako vyhledávací kritérium).

c) `Group(String name, Long quote, Integer gid, Boolean inGroup, String typeFs)` – vytvoření objektu skupiny s danými atributy.

3.2.2.3. Objekt `FileSystem`

Objekt `FileSystem` reprezentuje specifické části souborového systému – např. domovské adresáře uživatelů. Reprezentace pouze jedním objektem je zjednodušení, protože v managementu se tato entita váže ještě na entitu fyzických souborových systémů.

Souborové systémy (ve smyslu částí fyzických souborových systémů) se používají pro tyto účely:

- domovské adresáře uživatelů
- SW balíky – připojované automaticky nebo explicitně
- projekty – speciální diskové prostory na žádost. Zde je žádoucí zavést specifické atributy pro podporu managementu projektů – datum předpokládaného ukončení projektu, vazba na skupinu, která má k projektu přístup...

Atributy objektu `FileSystem`:

- `String name` – název souborového systému
- `String typ` – typ souborového systému
- `Long quote` – maximální velikost souborového systému
- `Boolean project` – informace, zda se jedná o projekt
- `Date finish` – datum předpokládaného ukončení projektu
- `String group` – skupina, která má přístup ke projektu
- `String owner` – vlastník souborového systému

Konstruktory objektu `FileSystem`:

a) `FileSystem(String name, String type, String owner, Long quote, FileSystemInfo fsi)` – vytvoření objektu souborového systému nebo projektu.

b) `FileSystem(String name)` – vytvoření objektu souborového systému nebo projektu se zadáním pouze jména.

c) `FileSystem()` – vytvoření prázdného objektu souborového systému nebo projektu.

V hlavním konstruktoru je použit objekt typu `FileSystemInfo`. V tomto objektu jsou zapouzdřeny informace o projektu. Použití zvláštního pomocného objektu je zde pouze z dů-

vodu přehlednosti. Konstruktor objektu `FileSystemInfo`:

```
FileSystemInfo(Date finish, String group)
```

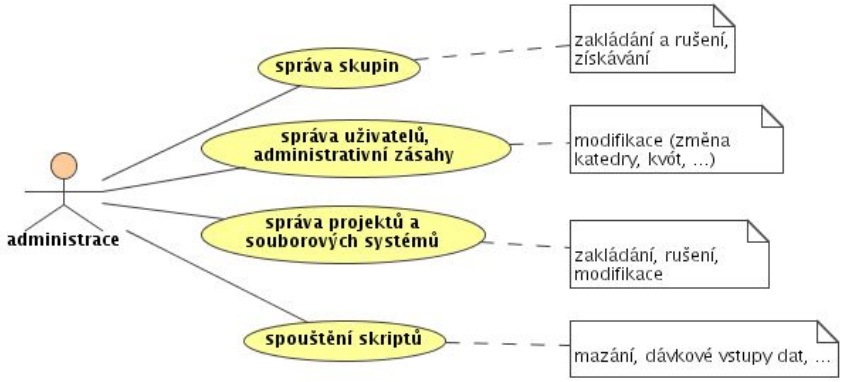
Není-li vytvářený souborový systém projekt, uvede se v konstruktoru `FileSystem` za `fsi` hodnota `null`.

3.2.3. Výkonné funkce aplikačního rozhraní

V následující podkapitole projdeme diagram užití a postupně navrhneme všechny potřebné funkce. Poznamenejme ještě, že řetězcové atributy v objektu kritérií vyhledávání budou vždy brány jako regulární výrazy.

3.2.3.1. Administrace

Navrhneme sadu funkcí týkající se administrace systému.

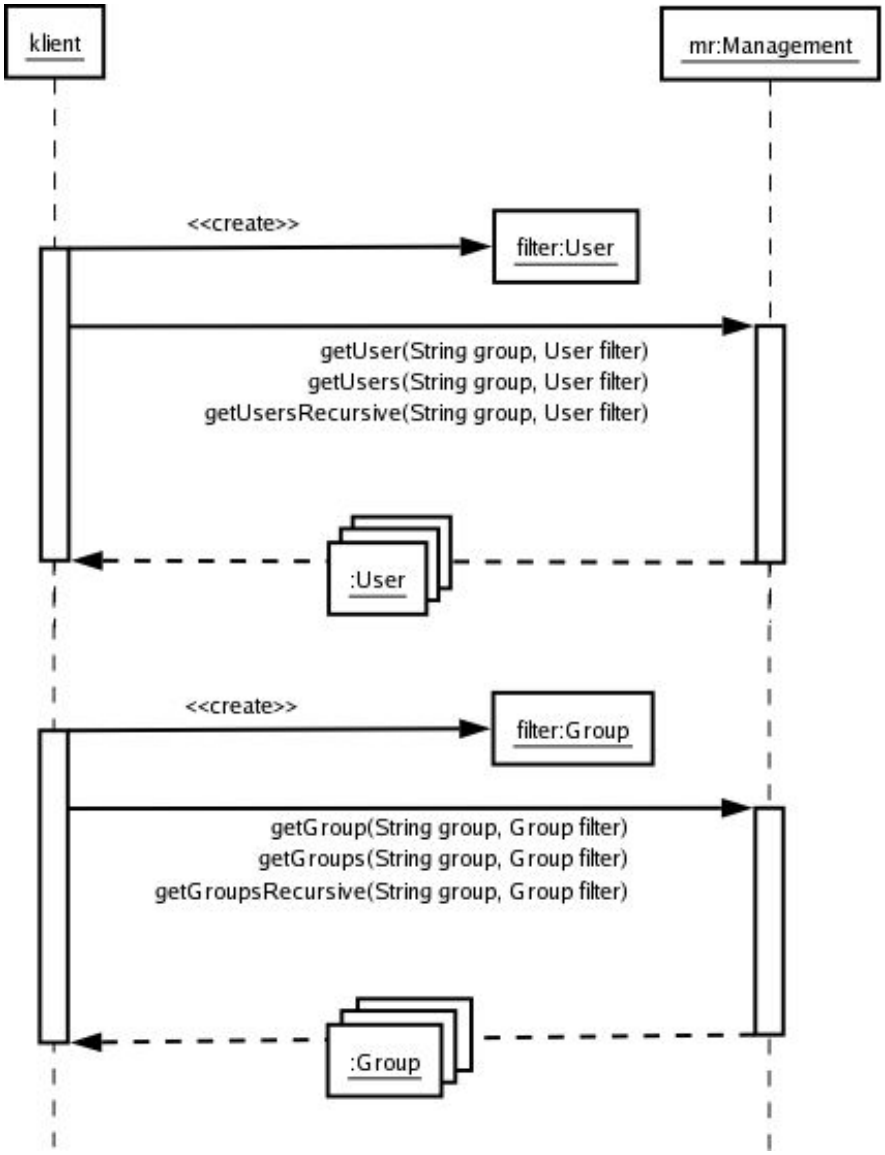


Obrázek 10: část diagramu užití – administrace

Správa skupin

Získávání informací o skupinách a uživatelích

Management poskytuje informace o jednotlivých skupinách nebo lze získat i množinu skupin nebo uživatelů ve skupinách podle jednoduchých kritérií:



Obrázek 11: volání metod pro získávání skupin a uživatelů

Objekt s kritériem ani jméno skupiny není povinné.

Popis jednotlivých funkcí:

- `ArrayList getGroups(String parent, Group criterium)` – vrací množinu skupin ve skupině `parent` odpovídajících zadanému kritériu. Strom skupin není procházen rekurzivně.
- `ArrayList getGroupsRecursive(String parent, Group criterium)` – vrací množinu skupin, které odpovídají atributům v objektu `criterium`. Strom skupin je procházen rekurzivně, vychází se ze skupiny `parent`.

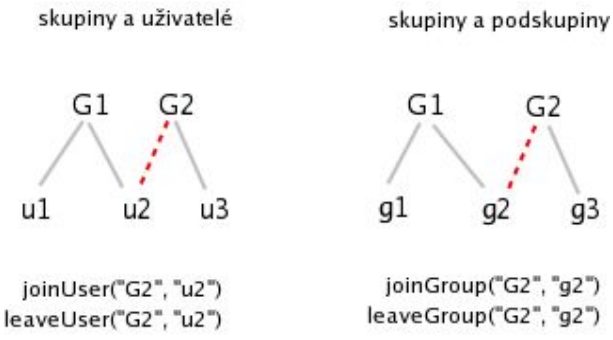
- `ArrayList getUsers(String group, User criterium)` – vrátí množinu uživatelů podle zadaného kritéria ve skupině `group`.
- `ArrayList getUsersRecursive(String group, User criterium)` – vrátí množinu uživatelů podle zadaného kritéria ve skupině `group`. Uživatelé budou získáváni rekurzivně.

Členství ve skupinách:

Strom skupin bude udržován následujícími funkcemi:

- `joinUser(String groupName, String login)` – do skupiny `groupName` se přidá uživatele `login`.
- `joinGroup(String groupName, String subGroup)` – do skupiny `groupName` se přidá podskupina `subGroup`.
- `leaveUser(String groupName, String login)` – odstraní se uživatel `login` ze skupiny `groupName`.
- `leaveGroup(String groupName, String subGroup)` – odstraní se podskupina `subGroup` ze skupiny `groupName`.

Příklad volání uvedených metod (červeně čárkovaně je vyznačeno přidávané/odebírané členství):



Obrázek 12: ukázka vkládání a odebíraní členů skupin

Modifikace skupin:

- `createGroup(Group group)` – vytvoří novou skupinu.
- `setGroup(String name, Group values)` – skupině se zadaným jménem nastaví nové atributy. Neuvedené atributy budou zneplatněny i v databázi (pokus o zneplatnění povinných atributů bude mít za následek chybu). Při změně pouze některých atributů je nutné

získat původní atributy voláním `getGroup()`.

- `modifyGroup(String name, Group values)` – modifikuje skupinu se zadaným jménem. Rozdíl oproti `setGroup()` jen ten, že tato funkce modifikuje pouze uvedené atributy (ne-`null` atributy).
- `removeGroup(String name)` – odstraní prázdnou skupinu. Nelze odstranit skupinu s podskupinami nebo s uživateli. Volající aplikace se musí postarat o jejich smazání.

Správa uživatelů

Modifikace kont:

- `setUser(String login, User values)` – uživatelskému kontu s daným jménem nastaví nové atributy. Neuvedené atributy budou zneplatněny i v databázi. Při změně pouze některých atributů je nutné získat původní atributy voláním `getUser()`.
- `modifyUser(String login, User values)` – uživatelskému kontu s daným jménem nastaví nové atributy, které jsou uvedené.
- `setStateUser(String login, boolean enable)` – povolí nebo zakáže uživatelské konto.
- `permitRemovingUser(String login)` – povolí se fyzické odebrání konta. Lze volat pouze na zablokovaném kontu. Je-li funkce volána na neinicializovaném kontu (stav 7), nevykoná se žádná akce ani chyba.

  Kromě realizace časového intervalu, kdy se nesmí konto smazat je takovýto způsob vhodný také proti nechtěnému smazání konta – konto musí být nejprve povoleno ke smazání touto funkcí.
- `removeUser(String login)` – odstraní uživatelské konto. Před odstraněním musí být odstranění povoleno voláním funkce `permitRemovingUser()` na zakázaném kontu.

Správa projektů a souborových systémů

Projekty, souborové systémy a domácí adresáře uživatelů jsou reprezentovány stejným objektem. Projekt má navíc atributy `finish` (datum předpokládaného ukončení projektu) a nepovinně `group` (skupina příslušející projektu).

Získávání informací o projektech:

- `FileSystem getFileSystem(FileSystem criterium)` – získání objektu soubo-

rového systému podle kritérií v `criterium`.

- `ArrayList getFileSystems(FileSystem criterium)` – získání množiny souborových systémů odpovídajících kritériu `criterium`.

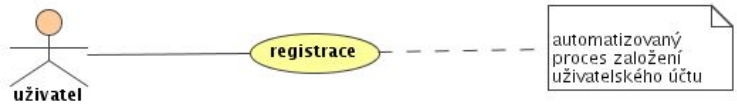
Zakládání, rušení a modifikace projektů a domovských adresářů:

- `createFileSystem(FileSystem fs)` – přidá souborový systém nebo domovský adresář. V objektu je několik povinných atributů: majitel, název, typ a kvóta.
- `removeFileSystem(String name)` – odstraní souborový systém nebo domovský adresář.
- `setFileSystem(String name, FileSystem fs)` – nastavení nových atributů.
- `modifyFileSystem(String name, FileSystem fs)` – modifikace atributů uvedených v objektu `fs` na požadované hodnoty.

Administrační skripty

Dávkové vstupy dat a mazání dat je možno realizovat již navrženými funkcemi.

3.2.3.2. Registrace kont



Obrázek 13: diagram užití – registrace

Registrace je vytvoření jednoho konta dané osobě. Aby bylo možno vytvořit uživatelské konto, musí být k dispozici jednoznačný nepredikovatelný identifikátor osoby – PIN.

Proces registrace je automatizovaný proces a má dvě fáze:

1) rezervování uživatelského jména a vytvoření položky v databázi
2) dokončení registrace a aktivace účtu

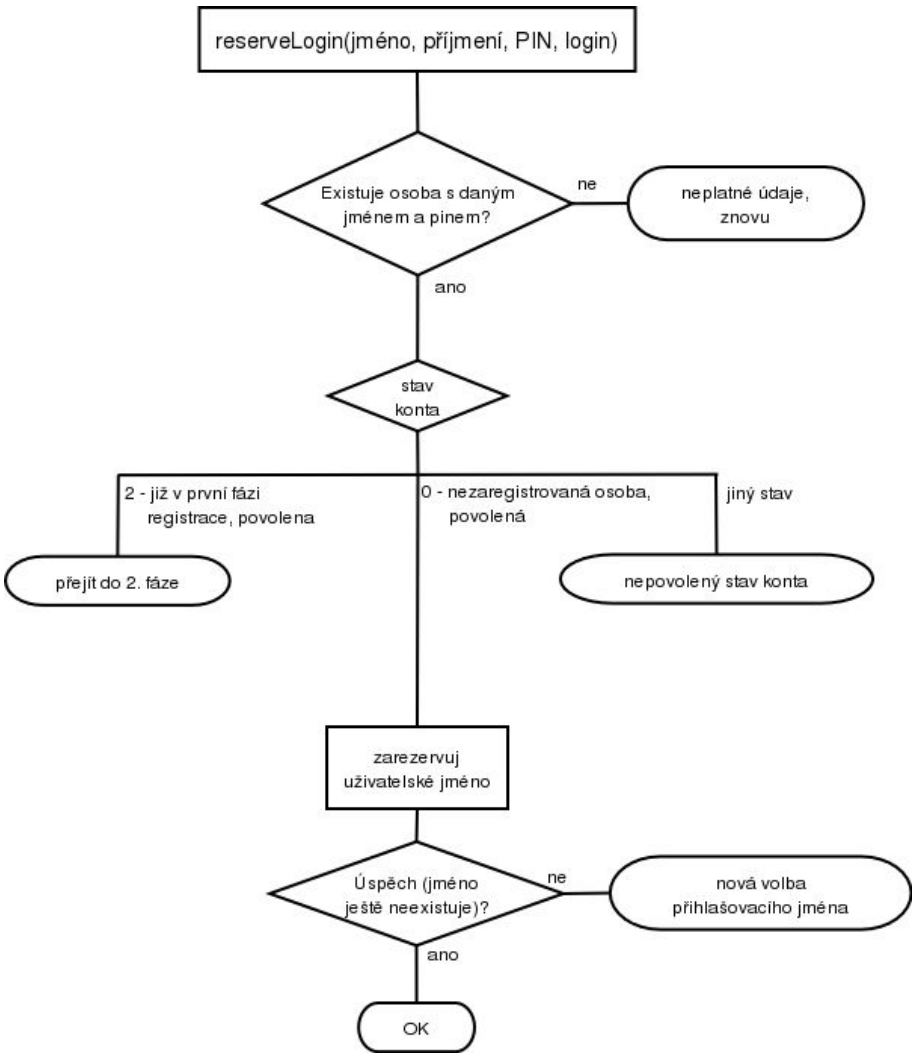
Uživatel smí zůstat v mezistavu.

Navržené rozhraní aplikační logiky:

- `reserveLogin(String firstName, String lastName, long PIN, String login)` – první část registrace – rezervování uživatelského jména.
- `finalizeReg(String login)` – dokončení registrace, konto se stane platným.

O vytváření, případně odesílání hesel se postará registrující aplikace.

Podívejme se blíže na první fázi registrace – rezervaci uživatelského jména. Funkce `reserveLogin()` musí provádět sadu kontrol. Algoritmus je vyznačen na obrázku:

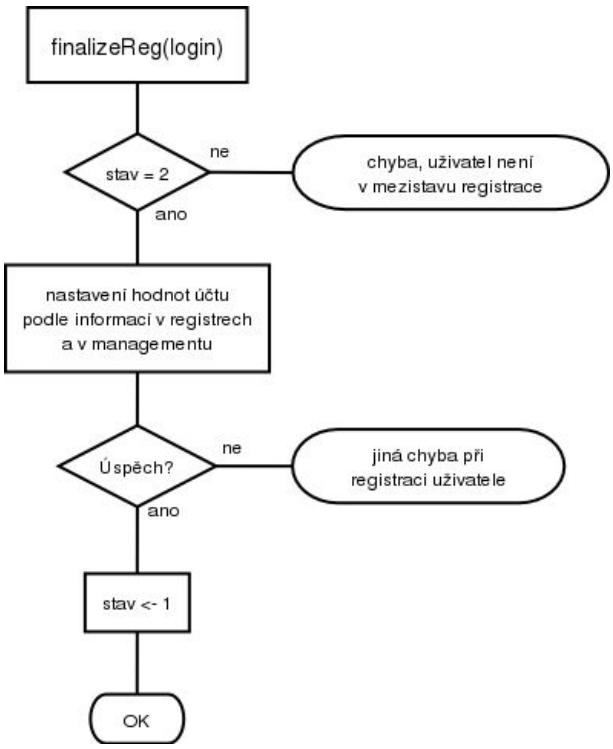


Obrázek 14: algoritmus funkce reserveLogin()

Nejprve se ověří existence osoby s daným registračním číslem PIN v databázi. Dále se musí zkontrolovat stav uživatelského konta. Zaregistrovat se pak smí pouze uživatel, který ještě není zaregistrovaný a není zakázaný (stav konta číslo 0).

V případě pokusu o registraci již napůl zaregistrovaného uživatele (v mezistavu 2) se rezervace uživatelského jména neprovede. Musí se přejít do 2. fáze registrace – volání `finalizeReg(login)`.

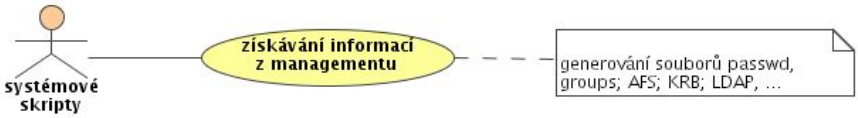
V druhé části registrace je rozhodování jednodušší, protože je cílem „jen“ dokončit registraci. Nejdůležitější je ověřit platnost a stav uživatelského účtu:



Obrázek 15: algoritmus funkce finalizeReg()

Vlastní dokončení registrace bude složitější operace závislá na konkrétních požadavcích kladených na management. Hlavně půjde o automatické nastavení některých hodnot uživatelského účtu buď z připravené šablony, nebo z výchozí skupiny odpovídající typu registrované osoby, a o nastavení dalších údajů, jako je např. UNIXovské UID.

3.2.3.3. Systémové skripty



Obrázek 16: diagram užití – spouštění systémových skriptů

Primární účel systémových skriptů je generování různých konfiguračních souborů. V této diplomové práci byl navržen management distribuovaného výpočetního systému pro

generování souborů `/etc/passwd` a `/etc/groups`.

Lze si představit realizaci této funkce i uvnitř managementu, ale již navržené funkce na získávání dat o uživatelích a skupinách by měly obecně postačovat pro celou škálu podobných funkcí.

3.2.3.4. Propagace změn z centrálního registru

Obrázek 17: diagram užití – synchronizace s centrálním registrem

Osoby jsou buď v kopii centrálního registru nebo pouze v lokální databázi managementu. Potřebujeme funkce pro přidávání, rušení a modifikaci osob. Funkce pro správu osob v kopii centrálního registru budou volány při synchronizaci centrálního registru informačního systému s kopií centrálního registru v databázi managementu.

Přidávání osob:

- `long addPeopleLocal(String firstName, String lastName)` – přidání nové osoby do lokálního registru. Funkce vrací osobní identifikační číslo, jehož rozsah se nepřekrývá s rozsahem osobního identifikačního čísla v centrálním registru.
- `addPeopleCentral(String firstName, String lastName, InfoStudent is, InfoEmploy ie, long PIN, long idReg)` – přidání osoby do lokální kopie centrálního registru. Tuto funkci lze použít např. pro synchronizaci centrálního registru s managementem.

Z důvodu přehlednosti jsou argumenty zapouzdřeny do pomocných objektů. Jejich konstruktory jsou:

- `InfoStudent(String SIN, boolean studying)`
- `InfoEmploy(String EIN, boolean employed, String phone, String room)`

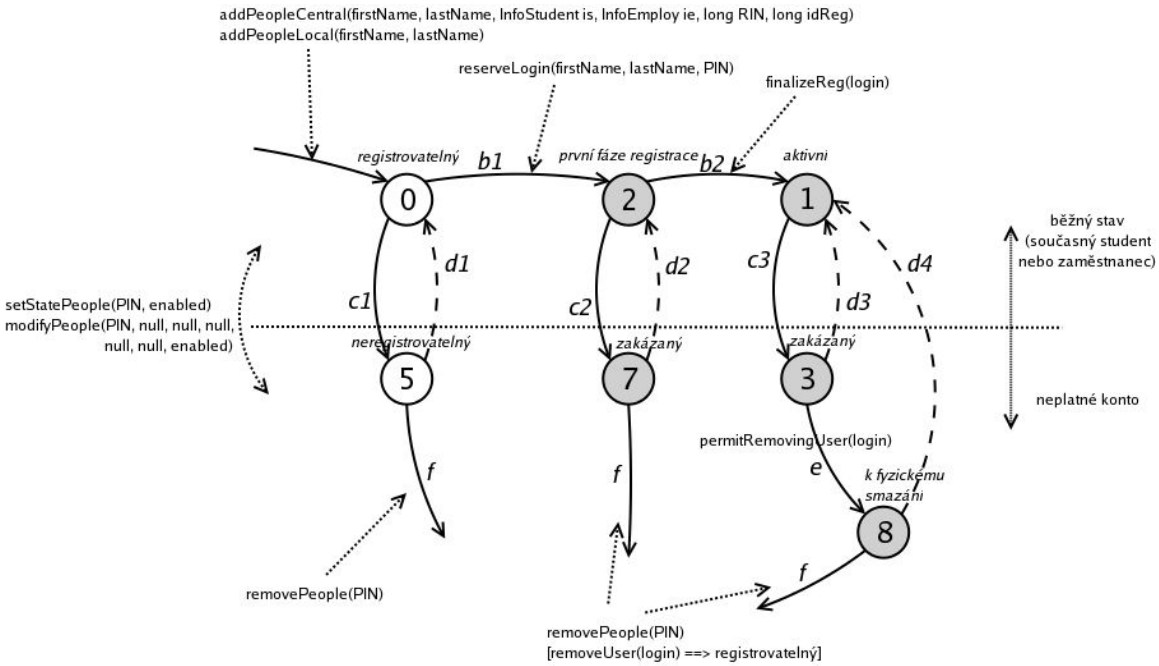
Modifikace a rušení osob:

Osoby se v systému identifikují pomocí osobního čísla. Toto číslo je jednoznačné v rámci centrálního i lokálního registru, proto budou používání společné funkce.

3. Návrh rozhraní aplikační logiky managementu DVP

- `void modifyPeople(long PIN, String firstName, String lastName, InfoStudent is, InfoEmploy ie, Long idReg, Boolean enabled)` – modifikace osoby v lokálním registru managementu nebo v kopii centrálního registru managementu
- `void setStatePeople(long PIN, boolean enabled)` – povolení nebo zakázání osoby a jejích uživatelských účtů. Ekvivalentní volání `modifyPeople(long PIN, null, null, null, null, null, null, new Boolean(enabled))`
- `void removePeople(long PIN)` – odstranění osoby z lokálního registru managementu nebo z kopie centrálního registru

Nyní lze nakreslit kompletní stavový diagram uživatelských kont s vyznačením volání funkcí aplikační logiky:



Obrázek 18: stavový diagram uživatele s vyznačením volání metod aplikačního rozhraní

4. Návrh a realizace ověřovací implementace

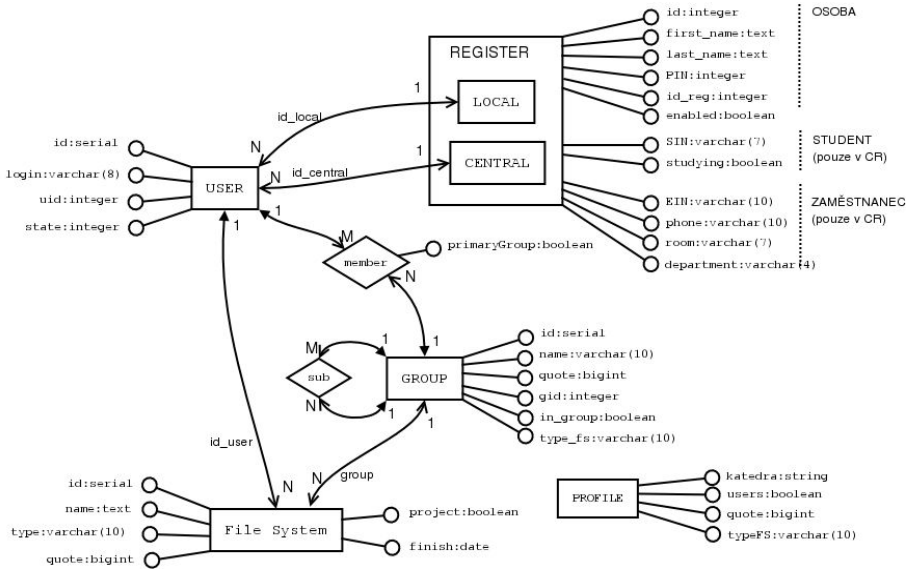
Použitelnost navrženého aplikačního rozhraní je ověřena na částečné testovací implementaci, datová vrstva byl implementována kompletně.

4.1. Datová vrstva

4.1.1. Datový model

Bylo nutné vytvořit datový model, nad kterým bude sada navrhovaných funkcí aplikační logiky pracovat.

Na datovou vrstvu budeme nahlížet tímto ERA modelem:



Obrázek 19: ERA model datové vrstvy ověřovací implementace

U tabulky `REGISTER` jsou vyznačeny skupiny atributů, kterým se zde říká *pozice*: „osoba“, „student“ a „zaměstnanec“. Pozice „osoba“ obsahuje základní informace, jako jsou jméno a příjmení a jednoznačný identifikátor osoby – PIN. Tabulka `REGISTER` má dvě části: osoby bez vazby na centrální registr osob (`LOCAL`) a osoby, které jsou v centrálním registru osob (`CENTRAL`).

V tabulce `CENTRAL` jsou přítomny všechny tři pozice – „osoba“, „student“ a „zaměstnanec“, v tabulce `LOCAL` je pouze pozice „osoba“.

4.1.2. Implementace datové vrstvy

Pro realizaci datové vrstvy je vhodný libovolný spolehlivý databázový systém s podporou kontroly dat, triggerů a transakcí. V této diplomové práci byl použit databázový systém PostgreSQL. Tento systém je relativně malý, robusní a volně šiřitelný pod licencí LGPL a

bývá standardní součástí většiny distribucí OS GNU/Linuxu, portován je také na další OS, např. Windows.

Komunikace s databází je možná pomocí javovské technologie Entity Beans nebo pomocí JDBC. Obě technologie se liší filozofií návrhu.

Technologie JDBC:

- výhodou je jednoduché rozhraní
- nevýhodou je nutnost ručně generovat řetězce s SQL příkazy, což by mohlo vézt i k bezpečnostním rizikům.

Technologie Entity Beans:

- složitější, vyžaduje více konfigurování
- výsledkem by ale mohlo být elegantnější propojení s prostředím J2EE

Pro implementaci aplikační logiky bylo nakonec vybráno rozhraní JDBC.

4.2. Aplikační logika

4.2.1. Typ java bean

Jak bylo zmíněno v teoretické části javovské beany se dělí do těchto skupin:

- Entity Enterprise Java Bean
- Message-driven Enterprise java bean
- Session Enterprise Java Bean

V této diplomové práci je použito pouze Session EJB, které poskytuje metody aplikačního rozhraní.

Session EJB se dále dělí na:

- Bezstavové – EJB nemá stav pro každé spojení, aplikační server může podle toho optimalizovat chování a používat jedinou instanci EJB.
- Stavové – EJB si pamatuje stav pro každé aktivní spojení. Toto by umožňovalo realizaci

složitějších operací – volání vícero souvisejících funkcí.

Základní otázka tedy je, zda použít bezstavový EJB nebo stavový EJB.

Zvolen byl bezstavový session EJB. Management je na to po vzoru Moiry připraven už na datové úrovni (např. stavy uživatelských kont). Bezstavový session EJB je také vhodný pro koncové webové aplikace, kdy se každá akce vykoná odesláním jednoho webového formuláře.

4.2.2. Nevýhody práce s Java Beans

Rozhraní funkcí na serveru a na klientu se liší – na klientu je u každé funkce navíc výjimka `RemoteException`. Nelze proto definovat společné rozhraní pro server i klient a zajistit tak transparentnost rozhraní už na úrovni jazyka Java. Lze sestavit server a klient s odlišným rozhraním a kontrola je prováděna automaticky až za běhu aplikace – při vyvolání neexistující metody je vrácena výjimka.

Grafický „deployovací“ nástroj hlídá platnost rozhraní při rozvinutí aplikace. Lze v něm také kontrolu provést explicitně („Tools“ --> „Verify J2EE Compliance...“).

4.3. Prezentační vrstva

Jako prezentační vrstvu si lze představit aplikační klient v Javě či jiném programovacím jazyku, servlet, portál a další. Klienty mohou být různé, ale všechny používají stejné aplikační rozhraní.

Prezentační vrstva v této diplomové práci slouží pouze k ověření funkčnosti aplikačního rozhraní. Implementované funkce aplikačního rozhraní byly ověřovány na konzolovém aplikačním klientovi v Javě. Implementován byl také jednoduchý servlet, kde stačí na klientovi pouze www prohlížeč.

4.3.1. Příklad použití navrženého aplikačního rozhraní

Různé typy klientů se v použití aplikačního rozhraní managementu téměř neliší.

Nejprve je třeba připojit se k serveru a získat spojku komponenty managementu:

```
Context cnt;
Object obj;
ManagementHome mh;
```

```
ManagementRemote mr;

cnt = new InitialContext();
obj = cnt.lookup("java:comp/env/ejb/ManagementFifne");
mh = (ManagementHome)PortableRemoteObject.narrow(obj,
ManagementHome.class);
mr = mh.create();
```

Další komunikace s managementem bude vykonávána pomocí objektu `mr`.

Příklad získání všech uživatelů s příjmením na „a“ ve skupině „users“:

```
User filter = new User(null, „A.*“, null, null, null, null);
ArrayList users = mr.getUsers(„users“, filter);
```

Příklad získání všech skupin:

```
ArrayList groups = mr.getGroups(null, null);
```

4.4. Praktické experimenty

4.4.1. Realizace objektů managementu v jazyce Java

Některé atributy v objektech managementu nejsou povinné, proto lze konstruktorům objektů předávat `null` namísto hodnoty. Korektnost musí kontrolovat aplikační logika a databáze.

Množství objektů, se kterými se v paměti pracuje může být velmi vysoké. Proto je otázka, jakým způsobem realizovat atributy získané z databáze, které nejsou povinné. Jsou zde dvě možnosti, jak atributy v objektech managementu realizovat:

- jako samostatný objekt: Integer, Long, … Hodnota `null` by znamenala „neuvedeno“.
- jako dva atributy: jeden s hodnotou (`int`, `long`, …), druhý s informací „platnosti“ (`boolean`)

První způsob vypadá logicky, elegantně zapadá do objektového modelu a je jednodušší. Druhý způsob vyžaduje v paměti méně tříd a tudíž by mohl být hardwarově méně náročný.

Při experimentování jsem zjistil, že druhý způsob je o polovinu méně paměťově náročný.

Zvolena byla ne o moc náročnější reprezentace pomocí objektů.

4.4.2. Praktická realizace

Práce byla odlaďována v tomto prostředí:

- operační systém GNU/Linux (Fedora Core 1 a 2), klient také pod Windows XP
- J2EE verze 1.4-beta2 a 1.4.2_02-b03
- databázový server PostgreSQL verze 7.3.4 a 7.4.2

Odzkoušeny byly tyto konfigurace:

a) server a aplikační klient běžící na jednom stroji

b) server a aplikační klient běžící na oddělených strojích v síti

c) server se servletem a vzdálený klient s www prohlížečem

4.4.3. Spuštění aplikace

Nejprve je nutné připravit server. Hotová aplikace je ve formě souboru `management.ear` (tento soubor vznikl předchozím „rozvinutím“ např. pomocí nástroje deploytool). Abychom mohli aplikaci použít, musíme ji rozvinout v našem aplikačním serveru. Rozvinutí provedeme pomocí příkazu:

```
asadmin deploy --user admin --retrieve bean bean/management.ear
```

Vznikne nám i aplikační klient (závislý na použitém aplikačním serveru), soubor `managementClient.jar`.

Dále zbývá spustit aplikačního klienta. Pouštění aplikačního klienta přímo na serveru J2EE nebo v síti se neliší. Chceme-li však pouštět vzdáleného klienta, budeme potřebovat kromě JAVA Runtime Environment i pomocné utility a třídy pro komunikaci se serverem – Application Client Container již zmíněný v teoretické části.

Vzdáleného klienta lze připravit takto:

1) Spustíme `package-appclient` z J2EE pod stejným typem OS, pod jakým poběží klient (TODO). Vznikne soubor `applient.jar`.
2) `applient.jar` přesuneme na klient a rozbalíme.
3) Přizpůsobíme konfiguraci v souboru `appclient/config/sun-acc.xml`: je třeba nastavit správné jméno vzdáleného serveru.

Dále může být nutné upravit spouštěcí skript `appclient/bin/alppclient[.bat]`.

4) Na serveru povolíme přístup klientů k nastavenému TCP portu, např.:

```
iptables -I INPUT -p tcp -m tcp --dport 3700 -j ACCEPT
```

5) klient spustíme příkazem:

```
appclient -client managementClinet.jar
```

Závěr

Výsledkem diplomové práce je analýza požadovaného chování managementu distribuovaného výpočetního prostředí na ZČU a návrh aplikačního rozhraní. Výsledné aplikační rozhraní představuje objekty a metody, které poskytuje jedna komponenta EJB (bean). Toto rozhraní bylo ověřeno na implementaci nejdůležitějších metod. Dále jsem vytvořil testovací prezentační vrstvu - aplikaci v Javě ovládanou interaktivně z konzole a servlet generující jednoduchou html stránku.

Protože je management DVP pro účely diplomové práce příliš rozsáhlá úloha, byly už během zkoumání funkcí managementu odstraněny z úlohy části, které jsou relativně nezávislé – správa strojů a sítí. Výsledné aplikační rozhraní managementu pak odpovídá této představě.

Aplikační rozhraní obsahuje pouze základní vyhledávací kritéria – rovnost atributů. Bylo by zřejmě možné napsat univerzálnější řešení, např. vnořované konstruktory objektů, řetězce připomínající SQL dotazy nebo xml řetězce, atd. Otázkou je, zda to má smysl. Složitější prohledávání může být řešeno buď na straně aplikace (a v kombinaci s použitím základního filtrování podle atributů) anebo může být aplikační rozhraní rozšířeno o specializované vyhledávací funkce. Myslím tedy, že cíle diplomové práce byly splněny. Kromě základního javovského klienta testující vybranou část aplikační logiky jsem vytvořil i jednoduchý servlet, který se automaticky nahraje při rozvinutí celé aplikace na aplikační server.

Za zmínku stojí použitý nástroj na sestavení. Typicky se v projektech s Javou používá `ant`. V diplomové práci jsem ale použil nástroj `make`. Kromě dobré znalosti psaní souborů pro `make` jsem si tento nástroj vybral také proto, abych se naučil přesnou práci s J2EE serverem: konkrétní příkazy nutné pro sestavení výsledné aplikace, nahrání ("rozvinutí") aplikace na aplikační server, pouštění aplikačního serveru, atd.

Důležité budoucí rozšíření je bezpečnost. Na nastavení přístupových práv by měla stačit konfigurace v deskriptorech pro rozvinutí managementu na server. Podle použitého

aplikačního serveru by pak byl dalším krokem buď zásah do kódu metod managementu DVP, nebo jen nastavení práv pro jednotlivé metody. Vlastní aplikační rozhraní by zůstalo beze zmeny.

Dalším vylepšením by mohlo být použití Entity Beans. (TODO)

Přehled zkratek

AFS.....Andrew File System (distribuovaný souborový systém)

API.....Application Program Interface (programové rozhraní)

COM.....Component Object Model

CORBA....Common Object Request Broker Architecture

DNS.....Domain Name Service (doménová jmenné služby)

DVP.....distribuované výpočetní prostředí

EJB.....Enterprise Java Bean (distribuované komponenty)

GIOP.....General Inter-ORB Protocol (specifikace komunikace mezi objekty ORB)

IIOP.....Internet Inter-ORB Protocol (implementace protokolu podle GIOP v TCP/IP)

J2EE.....Java 2 Enterprise Edition (platforma Javy pro distribuované aplikace)

JDBC.....Java Database Connectivity (přístup k databázi pomocí SQL v Jazyce Java)

JNDI.....Java Naming and Directory Services (jménné a adresářové služby)

JRE.....Java Runtime Environment

JSP.....Java Server Pages (dynamicky generované stránky)

KRB.....Kerberos (systém na ověřování uživatelů a služeb v síti)

LDAP.....Lightweight Directory Access Protocol (klient-server protokol adresářových a jmenných služeb)

MDVP.....management distribuovaného výpočetního prostředí

NDS.....Novell Directory Services (adresářové služby a správa uživatelů)

NIS.....Network Information Service (síťové vyhledávací služby)

ORB.....Object Request Broker (middleware obsluhující jednotlivé požadavky)

RMI.....Remote Method Invocation (vzdálené volání metod)

RMI-IIOP.vzdálené volání metod protokolem IIOP

SLP.....Service Location Protocol (konfigurace a hledání síťových služeb)

SPI.....Service Provider Interface (rozhraní pro zprostředkovatele – ovladač – JNDI)

Literatura

[sit] Jiří Sitera – projekt Pleiades
pracovní dokumenty pro management výpočetního distribuovaného prostředí
URL: http://home.zcu.cz/Pleiades/

[wie] Stephan Wieser – Java Enterprise Bean Tutorial
praktický popis a příklady použití EJB (pro javu ve verzi 1.3).
URL: http://rzserv2.fhnon.de/~lg002556/j2ee/ (odkaz už není platný)

[bona] Gary Bollinger, Bharathi Natarajan – Java server pages
průvodce návrhem JSP stránek

[paul] Paul J. Perrone – J2EE developer's handbook
podrobný průvodce J2EE

[sun] Java 2 Platform Enterprise Edition Specification, v1.4
dokumentace k implementaci J2EE od Sunu
URL: http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

[rom] Ed Roman – Mastering Enterprise JavaBeans, 2. edition
vysvětlení JavaBeans a souvisejících J2EE technologií
URL: http://www.theserverside.com/books/wiley/masteringEJB/