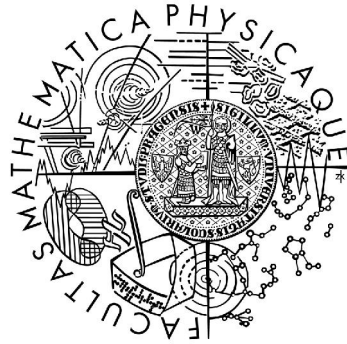


Univerzita Karlova v Praze  
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE



Dan Kobr

## **Systém pro skupinové plánování**

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.

Studijní program: Informatika, obecná informatika

2010

Děkuji vedoucímu bakalářské práce RNDr. Michalu Kopeckému, Ph.D. za cenné rady, připomínky a metodické vedení práce.

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

V Praze dne 26. května 2010

Dan Kobr

# Obsah

1	Úvod .....	1
2	Analýza.....	3
2.1	Klasifikace groupware aplikací.....	3
2.2	Systémy založené na internetových technologiích.....	4
2.3	Softwarové systémy s tlustým klientem .....	4
2.4	Charakteristika vybraných produktů.....	5
2.4.1	IBM Lotus Notes and Domino .....	5
2.4.2	Microsoft Exchange Server .....	6
2.4.3	Kontakt.....	7
2.4.4	Další alternativy aplikací s tlustým klientem .....	8
2.4.5	Produkty založené na internetových technologiích.....	8
2.5	Shrnutí .....	10
3	Specifikace .....	12
3.1	Celková koncepce aplikace .....	12
3.2	Definice a charakteristika částí aplikace .....	14
3.2.1	Charakteristika projektů .....	14
3.2.2	Základní hierarchie uživatelských skupin .....	15
3.2.3	Správa projektů .....	16
3.2.4	Správa uživatelských skupin.....	17
3.2.5	Kontakty.....	18
3.2.6	Kalendář .....	18
3.2.7	Žurnál.....	19
3.3	Závěrečné srovnání funkcí.....	20
4	Návrh aplikace.....	21
4.1	Problematika kompatibility se serverem Kolab.....	22
4.2	Serverová část.....	23
4.2.1	Dekompozice návrhu serveru .....	24
4.2.2	Formát dat přenášených mezi serverem a klientem .....	27
4.3	Klientská část .....	28
4.3.1	Dekompozice návrhu .....	28
4.3.2	Rozhraní pro komunikaci s jádrem.....	30

4.3.3	Rozhraní pro komunikaci s moduly .....	30
4.3.4	Vrstva poskytování dat .....	30
4.3.5	Vrstva přístupu k datům .....	30
4.3.6	Komunikace se serverem .....	31
4.3.7	Shrnutí.....	31
4.4	Návrh databáze .....	32
4.4.1	Hlavní entity .....	34
4.4.2	Doplňkové entity .....	35
5	Implementace .....	36
5.1	Datové objekty.....	36
5.1.1	Přehled tříd, jejich položek, vlastností a metod.....	37
5.2	Aplikační server.....	40
5.2.1	Třída DataAccessLayer .....	40
5.2.2	Třída CommunicationLayer.....	42
5.2.3	Zpracování dat.....	43
5.3	Klientská část .....	45
5.3.2	Knihovna PluginInterface.....	47
5.3.3	Jádro programu .....	49
5.3.4	Moduly .....	53
5.3.5	Zpracování dat.....	56
6	Závěr .....	59
7	Literatura .....	61
A.	Uživatelská příručka.....	62
	Instalace programu.....	62
	Přihlášení.....	64
	Hlavní nabídka .....	65
	Nastavení .....	66
	Změna uživatelských údajů .....	67
	Kalendář .....	67
	Kontakty .....	69
	Žurnál .....	71
	Projekty .....	71
	Pracovní skupiny.....	74

Příklad administrace pracovní skupiny .....	76
B.    Konfigurace .....	79
Instalace a nastavení aplikačního serveru .....	79
Ovládání aplikačního serveru .....	80
Konfigurace klientské části aplikace .....	81
C.    Obsah CD.....	82

## Seznam obrázků

Obrázek 1 - Schéma klient-server aplikace s databází .....	22
Obrázek 2 - Vrstvy serverové části aplikace a komunikace mezi nimi .....	24
Obrázek 3 - Návrh vrstev klientské části aplikace .....	32
Obrázek 4 - Zjednodušený diagram databáze bez vyznačených atributů .....	34
Obrázek 5 - Dědičnost tříd v knihovně DataObjects .....	37
Obrázek 6 - Průběh zpracování žádosti o data .....	44
Obrázek 7 - Přístup pluginů ke kolekcím dat .....	46
Obrázek 8 - Schéma tříd modulu Projects .....	56
Obrázek 9 - Zpracování žádosti o projekty .....	58
Obrázek 10 - Potvrzení instalace .....	62
Obrázek 11 - Výběr složky, do které se program nainstaluje .....	63
Obrázek 12 - Potvrzení instalace .....	63
Obrázek 13 - Dokončení instalace .....	64
Obrázek 14 - Přihlašovací okno programu.....	65
Obrázek 15 - Části hlavního okna .....	66
Obrázek 16 - Aktivace / deaktivace modulů .....	66
Obrázek 17 - Editace osobních údajů .....	67
Obrázek 18 - Kalendář .....	68
Obrázek 19 - Okno pro vytvoření nové události .....	68
Obrázek 20 - Událost zobrazená v kalendáři .....	69
Obrázek 21 - Kontextové menu položky kalendáře .....	69
Obrázek 22 - Kontakty .....	70
Obrázek 23 - Žurnál .....	71
Obrázek 24 - Projekty .....	72
Obrázek 25 - Okno pro vytvoření nového projektu .....	73
Obrázek 26 - Detail projektu.....	73
Obrázek 27 - Vytvoření nového úkolu .....	74
Obrázek 28 - Správa pracovních skupin.....	75
Obrázek 29 - Jedno z kontextových menu u administrace skupin.....	75

Obrázek 30 - Vytváření nové pracovní skupiny.....	76
Obrázek 31 - Vytváření nového uživatelského účtu.....	76
Obrázek 32 - Hierarchie uživatelů ve skupině.....	77
Obrázek 33 - Seznam projektů s nově vytvořeným projektem .....	77
Obrázek 34 - Hierarchie uživatelů skupiny po přidání nového projektu.....	77
Obrázek 35 - Přiřazování uživatele k projektu .....	78
Obrázek 36 - Uživatelé, kteří jsou přiřazeni k projektu .....	78

**Název práce:** Systém pro skupinové plánování

**Autor:** Dan Kobr

**Katedra (ústav):** Katedra softwarového inženýrství

**Vedoucí bakalářské práce:** RNDr. Michal Kopecký, Ph.D.

**e-mail vedoucího:** Michal.Kopecky@mff.cuni.cz

**Abstrakt:** Předložená práce se zabývá systémy pro skupinové plánování. Klasifikuje je do skupin a na základě analýzy vybraných existujících aplikací určuje funkce, které jsou pro tyto programy typické. Navržená a implementovaná groupware aplikace kombinuje jednoduchost a účelnost, které nabízí obdobné webové aplikace, spolu s komfortním uživatelským rozhraním a možností práce s daty bez on-line připojení, což jsou výhody aplikací s plnohodnotným klientem. Důraz byl kladen na snadnou použitelnost a modularitu, která umožní program jednoduše rozšiřovat, případně některé moduly deaktivovat a nahrazovat jejich funkčnost pomocí produktů třetích stran. Implementované moduly poskytují funkce pro práci s projekty, pracovními skupinami, kontakty, kalendářem a žurnálem.

**Klíčová slova:** groupware, správa pracovních skupin, skupinové plánování

**Title:** Collaborative Management Tool

**Author:** Dan Kobr

**Department:** Department of Software Engineering

**Supervisor:** RNDr. Michal Kopecký, Ph.D.

**Supervisor's e-mail address:** Michal.Kopecky@mff.cuni.cz

**Abstract:** This bachelor's thesis deals with collaborative management tools. It classifies them into groups and identifies typical functions of these programs. This thesis designs and implements new groupware application. Emphasis was put on its usefulness, ease of its use and modularity, which makes possible simple adding new functions and/or deactivating of existing modules and replacing their functionality with other third-party applications. Software combines simplicity of web based applications with comfort user interface provided by programs with fat client. Implemented modules allow administration of groups, administration of projects, calendar, contacts and journal.

**Keywords:** groupware, collaborative management tool, workgroup administration

# 1 Úvod

Není snadné definovat tak širokou oblast, jakou se groupware aplikace (nebo též *Collaborative System, Workgroups Support Systems*), česky *aplikace pro správu pracovních skupin*, zabývají. Lze říci, že *Groupware* zahrnuje komplexní software, který nabízí nástroje pro skupinové plánování, sdílení informací, komunikaci a týmovou spolupráci. Jedná se o programové vybavení, které nalézá uplatnění zejména ve firemním prostředí a umožňuje efektivní přístup k centrálně sdíleným informacím, které popisují činnosti vedoucí k realizaci veškerých cílů a zájmů, které si daný tým stanovil.

Ani toto vymezení není nijak zvlášť konkrétní. Otevírá se tak prostor pro velkou skupinu programů, z nichž každý naplňuje hlavní myšlenku skupinového plánování – umožnit efektivnější spolupráci prostřednictvím sdílení dokumentů a potřebných informací, ale liší se cílovou skupinou, pro kterou jsou určeny. Návrh a realizace takových aplikací pak může být diametrálně odlišná, neboť požadavky subjektů, které se rozhodly systém skupinového plánování používat, jsou nezřídka protichůdné.

Cílem této práce je navrhnout groupware aplikaci vhodnou zejména pro malé a střední firmy, případně jiné nezávislé skupiny, které kladou důraz na jednoduchost, účelnost a nízké pořizovací a provozní náklady. Software by přitom měl být dostatečně obecný a použitelný pro organizaci různě rozsáhlých projektů a pracovních skupin. S ohledem na to je nutné, aby software disponoval prostředky, potřebnými k modelování reálných situací – mám na mysli členění projektů na dílčí úkoly, definování závislostí mezi jednotlivými úkoly, přidělování úkolů konkrétním pracovníkům, nastavování přístupových práv uživatelům v rámci pracovní skupiny, aby mohl být omezen přístup ke zvoleným datům. Formulace zadání projektů a definice pracovních skupin by měly být jednoduché a přesné – takové, aby uživateli poskytly jasné instrukce k práci, včetně souvisejících materiálů, a aby uživatel nebyl nucen informace složitě dohledávat.

Následující kapitola analyzuje existující groupware systémy a snaží se porovnat jejich vlastnosti, funkční vybavenost a v neposlední řadě cenu. Na základě této analýzy jsou posléze stanoveny detailní cíle této práce.

Třetí kapitola se podrobněji zabývá požadavky, které byly vysloveny v předchozím rozboru. Rozděluje je na zásadní – ty, které nutně musí být součástí vyvíjené aplikace a rozšiřující, které budou implementovány s ohledem na časové možnosti při realizaci práce.

Čtvrtá kapitola popisuje návrh aplikace, který dané požadavky reflektuje. Na základě těchto a dalších kritérií rozděluje aplikaci na logické vrstvy, které od sebe budou v implementaci odděleny.

Následující kapitola je věnována reálné implementaci aplikace, popisuje nejdůležitější třídy a metody programu a na konkrétních příkladech ukazuje, jakým způsobem program zpracovává data.

Příloha obsahuje uživatelskou příručku s popisem instalace, konfigurace a práce s programem.

## 2 Analýza

Z úvodní kapitoly vyplývá, že skupina groupware aplikací je velmi rozsáhlá. Vzhledem k počtu různých systémů je vhodné produkty klasifikovat. Kritérií, která je možné aplikovat, je celá řada. V této kapitole jsou vybrána ta nejpodstatnější, od nichž se odvíjí podstata a návrh groupware aplikace. Členění je provedeno nejprve podle zvolené technologie, následně dle cílové skupiny, na kterou se groupware systémy mohou zaměřit. Závěr kapitoly se věnuje konkrétním produktům, které považuji za významné na poli programů pro skupinové plánování. U každého produktu je kromě charakteristiky a zařazení do konkrétních kategorií uvedeno také srovnání s navrhovanou aplikací.

### 2.1 Klasifikace groupware aplikací

Groupware je možné klasifikovat dle mnoha různých kritérií, v této kapitole budu software rozdělovat podle nejvýznamnějších dvou. V první řadě jde o celkovou koncepci programu. Programy tak dělím na systémy založené na internetových technologiích a na softwarové klienty s tlustým klientem. Bližší popis jednotlivých typů je zařazen v dalších kapitolách.

Druhé kritérium, které jsem zvolil, rozlišuje groupware produkty dle toho, do jaké míry integrují pracovní prostředí uživatele a odpovídají nárokům firem, které se rozhodly je používat. Zjednodušeně řečeno jde o dělení podle funkční vybavenosti. V následujících dvou odstavcích zavádím pojmy, které toto dělení dostatečně jasně vymezují.

*Collaborative Management Tools (CMT)* jsou aplikace, které zprostředkovávají zcela základní aktivity postačující ke spolupráci uživatelských skupin na konkrétních projektech. Typicky obsahují moduly pro správu projektů, kalendář, správu dokumentů a souborů, nějaký vhodný prostředek pro komunikaci uživatelů mezi sebou, případně nabízejí další funkce. Je zřejmé, že takový software představuje pro uživatele nástroj vhodný k organizaci pracovních týmů a poslouží jako pracovní diář, ovšem nejedná se o rozsáhlé a přitom komplexní prostředí, v němž by uživatel mohl realizovat veškerou práci, které se věnuje, včetně vytváření dokumentů či videokonferencí.

*Collaborative Project Management Tools (CPMT)* – CPMT vychází z CMT, ovšem značně tento termín rozšiřuje. Nenabízí pouze základní funkce spojené s organizací času a práce skupin, ale co se týče funkčního vybavení, zahrnuje všechny oblasti, které se týkají spolupráce a koordinace týmů. Mezi nejvýznamnější patří široké možnosti komunikace (videokonference, instant messaging, electronic meeting systems), pokročilé uchování dokumentů s ohledem na verze souborů, jejich sdílení nebo archivaci, a dále tzv. *knowledge management*, kam patří např.

web publishing, případně uchovávání informací na bázi modelu wiki. Je zřejmé, že takto mocný nástroj je velmi náročný na údržbu a správu a je vhodný zejména pro velké firmy, které si mohou dovolit jej financovat.

## **2.2 Systémy založené na internetových technologiích**

Nejpatrnější výhodou řešení, založených na internetových technologiích, je jejich nezávislost na platformě. Jsou přístupné pomocí libovolného internetového prohlížeče. Na druhou stranu právě webová prezentace značně svazuje návrhářům ruce. Taková řešení nedokážou nabídnout robustní pracovní prostředí, které se stane centrálním pracovištěm uživatele. Představují jen doplňkový nástroj, prostředek, se kterým uživatel interaguje jen tehdy, potřebuje-li zjistit nebo naopak vložit do systému informace, které potřebuje. Nevýhodou je tedy menší komfort takového rozhraní, které není srovnatelné s klasickou okenní aplikací. Z podstaty přístupu prostřednictvím webu dále plyne, že žádným rozumným způsobem nelze realizovat dostupnost potřebných dat offline, což může být často důvodem, proč se obrátit k aplikaci s tlustým klientem.

Pokud nejsou pro zákazníka jmenované nevýhody podstatné a aplikace je pro něj přijatelným řešením, jde o dobrou volbu, a to zejména z ekonomického hlediska. Cena webových groupware aplikací bývá mnohonásobně nižší než cena za pořízení aplikace s tlustým klientem, nehledě na to, že téměř odpadají problémy s kompatibilitou a není nutné se starat o aktualizaci a údržbu softwaru na klientských stanicích.

## **2.3 Softwarové systémy s tlustým klientem**

Jedná se o programy, které nejsou omezené internetovým rozhraním, mohou nabídnout patřičný komfort ovládání a dokážou pracovat i v offline režimu. Jsou závislé jen na platformě klientské části aplikace. Součástí nákladných řešení velkých firem je často přítomnost doplňkového webového rozhraní pro přístup k datům přímo z internetu. Aplikace tohoto typu dokážou tedy uživateli zprostředkovat potřebné informace odkudkoli bez ohledu na operační systém a nevýhodu této kategorie tak stírají. Nebývá to ovšem standardem, proto v této části analýzy nebudu počítat s tím, že produkty takovým doplňkem disponují.

Systémů s tlustým klientem je k dispozici velké množství, jak bylo řečeno výše, najdeme mezi nimi takové, které disponují nepřebornou škálou funkcí, možnostmi komunikace či synchronizace s dalšími aplikacemi, ale i ty, které se svou jednoduchostí blíží běžnému diáři, který pouze sdílí větší skupina uživatelů. Rozdíly jsou přirozeně také v ceně, která reflektuje nejen funkční vybavenost, ale také kvalitu technické podpory, rychlost řešení problémů či otevřenost dalšímu rozšiřování produktu.

## 2.4 Charakteristika vybraných produktů

Pro každou skupinu aplikací (viz klasifikace v kapitole 2.1) jsem vybral několik konkrétních produktů, které uvedené typy dobře reprezentují. Kromě popisu nabízených funkcí a technologií, na kterých jsou produkty založeny, se pokouším také určit, pro jaké zákazníky co do velikosti je software určený a jak ovlivňuje svým rozsahem a funkčností způsob práce uživatele. Rád bych se tak dostal ke shrnutí, ze kterého by mělo být zřejmé, jaké základní rysy bude splňovat aplikace, kterou navrhuji.

V prvních čtyřech bodech této kapitoly se zmíním o produktech ze skupiny softwaru s tlustým klientem. Jedná se totiž o programy, které na poli groupware patří mezi nejžádanější a nejpoužívanější. Poslední bod této kapitoly (2.4.5) shrnuje a rozebírá významné zástupce z řad webově orientovaných aplikací. Méně prostoru je tomuto typu programů věnováno proto, že jejich obecná charakteristika je velmi podobná a postačí, bude-li uvedena jen jednou. Liší se v podstatě pouze v nabízených funkcích, které jsou u každého produktu zmíněny na konci kapitoly.

### 2.4.1 IBM Lotus Notes and Domino

Jedná se o velmi propracovaný software, jeden ze dvou majoritních produktů na poli groupware aplikací. Z hlediska celkového pojetí přístupu k organizaci času, úkolů a pracovních týmů jde jednoznačně o CPMT aplikaci. Systém je natolik rozsáhlý a škálovatelný, že lze hovořit spíše o platformě pro vývoj groupware aplikací. Zvolená architektura je typu klient server. Klient je označován jako *IBM Lotus Notes*. Serverová aplikace nese název *Domino*.

**Technologie:** Komunikace mezi částmi aplikace probíhá pomocí nativního protokolu *NRPC*<sup>1</sup>, a to jak mezi klientem a serverem, tak mezi jednotlivými servery – je to z toho důvodu, že prostředí IBM Lotus Domino je distribuovaným systémem, přičemž servery jsou mezi sebou replikovány. Serverové aplikace představují distribuované databáze, mimo to servery Domino poskytují další standardní služby (POP3, HTTP, LDAP) a podporují mnoho standardů (XML, Java).

**Funkce:** V případě tohoto produktu prakticky neexistuje služba typická pro groupware aplikaci, kterou by nemohl nabídnout. Klientská část aplikace je sama o sobě (bez přístupu k serveru) propracovaným prostředím, které uživatel nebude nucen opouštět kvůli tomu, že by nějakou funkci postrádal. Ostatně funkcí je k dispozici skutečně nepřehledné množství – ze základních jsou to kalendář, e-mail, kontakty, nejrůznější přehledy aktivit, správa dokumentů, apod. Vše je možné sdílet, definovat přístup skupinám uživatelů, exportovat do standardních formátů, synchronizovat s jinými aplikacemi. Je vhodné zdůraznit, že výrobce tohoto softwaru si zakládá na bezpečnosti dat – ta je založena na existenci tzv. ID souborů s citlivými daty, jakými jsou například certifikáty, šifrovací klíče či digitální podpisy.

---

<sup>1</sup> *Network Remote Procedure Call* – vzdálené volání procedur (*network* zdůrazňuje, že jde o síťovou komunikaci); umožňuje programu volat proceduru umístěnou jinde než je volající program

Ve spojení s RSA šifrováním veřejným klíčem lze zabezpečení dat zajistit na mnoha úrovních.

**Shrnutí:** Rozhodnutí jakékoli větší či menší firmy investovat do produktu *IBM Lotus Notes and Domino* a aktivně jej využívat představuje radikální změnu v dosavadním přístupu k práci jednotlivých uživatelů a je dobré si takový krok důkladně rozmyslet. Software je totiž natolik dominantní a robustní, že je velice pravděpodobné, že daný podnik bude nucen přehodnotit a změnit stávající přístup k zadávání, realizaci a odevzdávání výsledků provedené práce zaměstnanci. Může se zdát, že vyřčené tvrzení je poněkud odvážné, ovšem je nutné si uvědomit, že jiný podobný software prozatím na trhu není k dispozici. Od uvedené charakteristiky se odvíjí bezpočet argumentů, proč používat tento software či ne a pro jaké korporace je vhodný.

Ukázalo se, že se *IBM Lotus Notes* snaží být natolik komplexní a postihnout tak širokou škálu funkcí, které by mohl uživatel využít, že původní myšlenka CPMT – totiž administrace projektů – poněkud zapadla. Jsou vyzdvihovány jiné funkce a přednosti programu a ani po delším prozkoumání jsem nenašel způsob, jak přesně definovat projekt, detailně rozdělit práci týmu a rozumně zadané projekty spravovat. Ba co víc – když jsem po informacích tohoto typu pátral, byl jsem odkázán na rozšiřující moduly tvůrců, kteří s původním producentem softwaru nemají co do činění. Netvrdím, že se jedná o aplikaci nevhodnou pro správu projektů a podobné využití, ale považuji za podstatné zdůraznit, že *IBM Lotus Notes* nenabízí tyto funkce natolik propracované, jak bych považoval za vhodné. Podotýkám, že většina níže popsaných aplikací věnuje samotné správě projektů pozornost o poznání větší.

Zakoupí-li firma toto programové vybavení, nedává tím svým zaměstnancům žádnou možnost volby. Jak je psáno výše – pracovní prostředí klienta *IBM Lotus Notes* je natolik komplexní, že je více méně nutné většinu práce vykonávat právě v něm. Za výtku stojí také způsob ovládání, který se vymyká standardům, na který jsou uživatelé zvyklí, a není příliš ergonomický.

Výhody, které plynou z investice do této aplikace, byly vesměs již jmenovány (za všechny jde například o modularitu, nepřebornou škálu nabízených funkcí, vyzkoušený software s dlouhou historií).

*IBM Lotus Notes and Domino* je programové vybavení, které ocení zejména velké firmy a korporace. Nevyplatí se jej kupovat, shání-li firma pouze pomocný nástroj pro organizaci práce týmů – proti hovoří zejména cena a fakt, že správa projektů už zdaleka není tím jediným, na co se program soustřeďuje.

## 2.4.2 Microsoft Exchange Server

Pro většinu uživatelů (či spíše managerů) je právě *MS Exchange Server* synonymem pro groupware aplikaci. Díky své pozici na trhu má Microsoft obrovskou výhodu, ovšem nejen to mu usnadňuje rozšiřování svého produktu. Zásadní je skutečnost, že výrobce integroval funkce Exchange Serveru do známého e-mailového klienta *MS Outlook*. Uvědomíme-li si, že *MS Outlook* je součástí

kancelářského balíku MS Office, je jasné, že Microsoft má výtečnou pozici pro distribuci služeb Exchange Serveru.

**Technologie:** Opět jde o aplikaci postavenou na architektuře klient server. Jednotlivé části komunikují pomocí protokolu RPC, který využíval již samotný *MS Outlook*. Výrobce klade důraz mimo jiné na možnosti synchronizace, kterých je celá řada – věnuje pozornost nejen aplikacím a systémům, které sám produkuje (např. pro mobilní zařízení), ale i dalším programům, které jsou na trhu. Minimálně stejná pozornost je věnována zajištění bezpečnosti a integrity dat.

*MS Exchange Server* je vhodným řešením pro větší podniky a korporace, které si mohou dovolit vynaložit nemalé prostředky nejen na zakoupení licence, ale také na správu serveru Exchange. Srdcem celého systému je známé *Active Directory*, které nabízí nepřehlednou škálu možností, jak celý systém konfigurovat. Zdůraznil bych propracovaný systém uživatelských rolí či pokročilé možnosti sdílení souborů.

**Funkce:** Ačkoli funkcí, které koncoví uživatelé a jejich nadřízení ocení, je celá řada (kalendářem, kontakty a úkoly počínaje, definováním rolí či přístupem z webového rozhraní konče), opět jsem při bližším prozkoumání softwaru nenalezl nástroj, který by umožnil exaktně definovat úkoly, rozdělit je podřízeným na dílčí části a spravovat celý přehled takových povinností. Z toho plyne velmi podobná charakteristika, jaká je uvedena u *IBM Lotus Notes and Domino*.

**Shrnutí:** Svým způsobem se *MS Exchange Server* stále více přibližuje produktu od IBM: S novějšími verzemi přichází další a další rozšíření – od letošního roku najdeme mezi funkcemi například VoIP telefonii, což je rys typický pro rozsáhlou CPMT aplikaci. A nejen to, výrobce své produkty dále sjednocuje. Výsledkem je *MS Office* s integrovanými službami *MS Exchange Serveru*, případně naopak. Zejména to by mělo být indicií, že si produkty od IBM a Microsoftu budou stále více podobné. Exchange Server rozhodně nemá ambice stát se subtilní aplikací typu klient server, využívanou uživateli jen tu a tam v případě, že potřebují zjistit, jak si počínat dál při realizaci nějakého projektu.

### 2.4.3 Kontakt

Jedná se o jediného správce osobních informací a groupware aplikaci, určenou ryze pro unixové systémy, který zde uvedu. Nástroj je samozřejmě plně grafický – využívá KDE<sup>2</sup>. Je to poslední program s tlustým softwarovým klientem v tomto přehledu. *Kontakt* je velmi jednoduše konfigurovatelný. Jde v podstatě o soubor aplikací – modulů, které může uživatel libovolně vypínat a zapínat podle potřeby. Za všechny jmenuji kalendář, správce kontaktů, e-mailového klienta a samozřejmě nástroj pro administraci projektů. Někteřími rysy je *Kontakt* podobný *MS Exchange serveru*. Klientem je nezávislá aplikace, která v mnoha případech slouží uživatelům sama o sobě, podobně jako *MS Outlook*. Teprve rozhodne-li se uživatel využít dalších služeb a funkcí, typických pro groupware, začne aplikace využívat služeb serveru, vyvinutého pro tyto účely.

---

<sup>2</sup> *K Desktop Environment* – desktopové prostředí pro Linux a jiné unixové systémy

**Technologie:** Nejdůležitějším prvkem je server – byl vyvíjen nezávisle pro účely groupware aplikací a nese název *Kolab*. *Kolab* server mohou využívat také jiní klienti. Zásadní myšlenkou je využití protokolu IMAP nejen pro posílání e-mailů, ale také pro přenos všech ostatních zpráv, potřebných pro funkčnost doplňků utvářejících groupware. Jedná se o konfigurační zprávy, které jsou ukládány do zvláštních IMAP složek ve stanoveném Kolab-XML formátu. Server se pak stará o přístupová práva k tomuto systému souborů.

**Funkce:** *Contact* je jednoduchá, přehledná aplikace, která nabízí velkou část funkcí, kterou by měl obsahovat každý groupware – kromě jmenovaných základních modulů stojí za zmínku sdílení kontaktů a událostí v kalendáři, šifrování dat či alternativní přístup skrz webové rozhraní. Nechybí ani možnost řadit uživatele do skupin a přiřazovat jim role. Nedostatkem, který by mohl být pro určité druhy projektů kritickým, je nemožnost sdílet soubory a asociovat je s konkrétními úkoly. Nakonec je vhodné zmínit, že *Contact* je poskytován jako open source (licence GPL), což představuje jeho konkurenční výhodu.

**Shrnutí:** Z popisu je patrné, že *Contact* nespadá do stejné kategorie jako předešlé aplikace. Akceptuje-li zákazník platformu, na které *Contact* běží, získá tak pro svoji – typicky menší – firmu velmi dobré a spolehlivé řešení. Uživatelé jistě ocení vysokou míru přizpůsobivosti softwaru a zejména jednoduchost a intuitivní ovládání. Jelikož jde na první pohled o groupware typu CMT, nelze očekávat, že by program nahradil aplikace nutné pro zajištění interní komunikace ve firmě či jiné správu nahrazující programy.

#### 2.4.4 Další alternativy aplikací s tlustým klientem

Samozřejmě existuje celá řada dalších groupware aplikací s nativními klienty, ovšem ty nejpoužitelnější z nich, které tu ještě nebyly jmenovány, jsou velmi podobné těm třem, diskutovaným výše. Jsou to například *Open XChange Server*, který představuje alternativu serveru *Kolab*. Dokáže tedy mimo jiné spolupracovat s uvedeným programem *Contact*. Oproti *Kolabu* umožňuje jednoduché sdílení souborů a jejich přiřazování k projektům. Je však na místě podotknout, že tato služba je, na rozdíl od *Kolabu*, placená.

Pro Linuxové servery lze doporučit také open source *Zimbra*. Opět se jedná o aplikaci velmi podobnou *Kolabu* či *XChange Serveru*, ovšem nenajdeme zde funkce, které jsou pro groupware aplikaci poměrně zásadní. Například rozdělení úkolu na dílčí části či vytváření uživatelských skupin.

#### 2.4.5 Produkty založené na internetových technologiích

Groupware aplikace, přístupné výhradně pomocí internetového prohlížeče, mají mnoho společných rysů – v zásadě se liší jen rozsahem nabízených funkcí a cenou. Většina z nich jsou open source projekty – najdeme mezi nimi takové, které nabízejí jen základní funkce, ale překvapí i produkty, které propracovaností dávají zapomenout na svůj nekomerční charakter.

Internetové groupware aplikace jsou typickými zástupci CMT. Není se čemu divit – jejich tvůrci jsou svazováni internetovým rozhraním, kterému je třeba se přizpůsobit. Z mého pohledu je to ku prospěchu věci. Tyto groupware systémy totiž vynikají jednoduchostí a přehledností. Jejich cílem není být komplexní software, který si klade za cíl vyjít vstříc v každém ohledu běžnému uživateli, jako tomu je u produktů společností IBM Lotus či Microsoft a dalších. Toto tvrzení jde do jisté míry zobecnit. Aplikace jsou vhodným řešením pro menší a střední firmy, které od groupware žádají pouze pomocný nástroj, který jim pomůže zpřehlednit a zorganizovat práci svých zaměstnanců.

**Technologie:** Technologie není nijak převratná. V drtivé většině případů se jedná o PHP ve spojení s databází MySQL, zejména díky tomu, že MySQL je na většině web-hostingových serverech dostupná zdarma. Ale ani použití ASP či JSP není výjimkou.

Nebudu se dlouze rozepisovat o konkrétních produktech, které jsou k dispozici, u vybraných pouze přehledně, v bodech zdůrazním základní či pokročilé funkce a vlastnosti daného softwaru. Nakonec bych pro úplnost rád dodal, že kromě níže uvedených aplikací stojí za vyzkoušení také *PHPgroupware* či *Outreach*.

#### 2.4.5.1 DOTProjekt

- open source
- **základní funkce:** projekty, úkoly, kalendář, správa souborů, kontakty, správa uživatelů a práv
- **rozšiřující funkce:** informace o připojených klientech, fórum
- **klady:** propracované uživatelské rozhraní, modularita, snadná lokalizace
- **zápory:** nenabízí možnost řadit uživatele do skupin

#### 2.4.5.2 Moregroupware

- **základní funkce:** projekty, úkoly, kalendář, kontakty, poznámky, správa uživatelů (vč. zařazování do skupin) a projektů
- **rozšiřující funkce:** novinky, široké možnosti osobního nastavení, webmail
- **klady:** velké množství nabízených modulů, propracované funkce, přehledné zobrazování náhledů (přehledů projektů), obsáhlá nápověda
- **zápory:** chybí správa a sdílení souborů
- **stručný popis:** Aplikace působí velmi příjemným a hlavně účelným dojmem. Jednotlivé moduly jsou do detailů propracované, najdeme tu mnohé funkce, kterými nedisponují ani programy komerčního ražení – za všechny bych jmenoval možnost exportovat téměř všechny přehledy do formátu PDF (např. kontakty), možnost nahlédnout do kalendářů ostatních uživatelů (vhodné při plánování schůzek) či stránka rešerší zpravodajských serverů, což uvádím spíše jen pro zajímavost.

### 2.4.5.3 PHP Project

- **základní funkce** – projekty, úkoly, kalendář, kontakty, poznámky, správa souborů
- **pokročilé funkce** – chat, diskusní fórum, možnost uživatelského přizpůsobení programu, webmail, přehled času stráveného prací, kontextově citlivá nápověda
- **klady** – plně modulární systém, funkčně velmi pokročilé
- **zápory** – velmi těžkopádný vzhled a nelogické rozmístění ovládacích prvků

## 2.5 Shrnutí

Je přirozené se ptát, zda dostupné groupware produkty pokrývají poptávku – ještě lépe, zda každý potenciální zákazník najde na trhu dostatečně vhodný produkt splňující většinu nároků, které si klade. Rozebral jsem zástupce aplikací několika kategorií a zmínil jsem jejich výhody a nevýhody. Nyní se pokusím tyto poznatky shrnout a určit, co by mělo být nedílnou součástí groupware, co je nadbytečné a co je vhodným rozšířením.

### Elementární funkční vybavení groupware aplikace:

- správa projektů – jasně definovatelné projekty a jejich dílčí části, přímočaré přiřazování uživatelů ke konkrétním úkolům, jednoduché určení oprávnění uživatele v rámci projektu či skupiny
- správa uživatelských skupin – realizované podobně jako správa projektů, tedy vytváření uživatelských skupin, rozřazování uživatelů do skupin, případně změna oprávnění uživatele v rámci dané pracovní skupiny
- kalendář – přehledné zobrazení blížících se termínů projektů a jejich částí, přidávání dalších událostí, které vytvoří uživatel
- kontakty – správce kontaktů, automaticky zahrnující kontakty z pracovních skupin uživatele s možností synchronizace například s e-mailovým adresářem, případně zálohy dat

### Rozšiřující moduly:

- sdílení souborů – tuto funkci nepovažuji za zcela nezbytnou, ovšem přijde mi více než vhodné, aby software pro distribuované rozdělování práce umožnil kupříkladu přiložit popis projektu místo toho, aby musel být ručně vepsaný. Každého jistě napadne mnoho dalších podobných situací (například soubory s grafikou), kde je sdílení využitelné. Ostatně ve specifikaci je tento požadavek uveden jako jeden ze základních.
- žurnál – rychlý komunikační kanál, sloužící především ke komentování zadání a řešení projektů

### **Doplňující funkce:**

- integrace běžně používaných komunikačních prostředků – e-mailový klient, diskusní fórum. Vytvoření e-mailového klienta je značně diskutabilní, dle mého názoru může ovlivnit pohled na koncept aplikace jako takový, totiž zda půjde o CMT nebo CPMT. E-mail je v dnešní době natolik používaný prostředek komunikace, že implementace zcela nového poštovního klienta by byl krok přinejmenším odvážný, kterému bych se rád vyhnul. Mimo jiné se domnívám, že groupware aplikace by neměla sloužit jako intenzivní obousměrný komunikační kanál, ale spíše jednosměrný. Základem je dobrá analýza projektu, rozdělení úkolů a poskytnutí dostatečného množství materiálů pro jeho realizaci. To vše poskytuje server klientovi (resp. definuje nadřazený podřízenému). Elektronická pošta je v této situaci pouze doplňkovým prostředkem pro upřesnění dohody, v některých případech může sloužit k odevzdání hotové práce. Na druhou stranu by vhodným doplňkem mohlo být zasílání upozornění na zvolený e-mail v případě, že dojde například k vytvoření nového úkolu, který musí daný uživatel vypracovat.
- podpora Kontaktových serverů – daleko lepší rozšiřitelnosti by bylo možné dosáhnout podporou některého z již zavedených a známých serverů. Tou nejlepší volbou se zdá být server Kolab, a to nejen díky otevřenému formátu. Je to zejména z toho důvodu, že Kolab využívá celá řada dalších klientů, včetně MS Outlook, pokud je vybaven vhodným pluginem. Z hlavních funkcí serveru stojí za zmínku možnost sdílení kontaktů a kalendáře, administrace prostřednictvím webového rozhraní či LDAP adresář pro uchovávání konfiguračních dat. Některé funkce podporovány nejsou, například sílení souborů by pomocí tohoto serveru možné realizovat nebylo.
- správa verzí – implementací modulu správy verzí by byl eliminován problém s využitím programu pro softwarové projekty, pro které je tato funkce podstatná. Je vhodné podotknout, že groupware aplikací podporujících správu verzí je poskrovnu, spíše se objevují samostatně jako doplňkové aplikace.
- pokročilé sdílení souborů – uživatel by mohl definovat sdílenou složku a pro konkrétní skupiny či jednotlivé uživatele a určovat, jak mohou se sdílenými soubory nakládat. Podobnou funkcí disponuje komunikátor Windows Messenger.
- Další přínos pro groupware aplikaci spočívá například ve zpřístupnění podnikových dokumentů, vytvoření znalostní báze (zejména pro vedoucí zaměstnance), případně v možnosti sdílení nástrojů a dalších pomocných souborů

### **Požadavky na zpracování aplikace a jednotlivých modulů:**

- modularita

- umožnění exportu a importu dat z vybraných modulů, kompatibilita těchto výstupů s dalšími programy

Dohromady nejsou požadavky nijak náročné. Aplikace, která by je splňovala, bude typickým zástupcem CMT groupware. Pokusím se zhodnotit, do jaké míry odpovídají výše popsané produkty těmto požadavkům. Až na výjimky je splňují programy webové (uvedené v kapitole 2.4.5). *IBM Lotus Notes and Domino* a *MS Exchange Server* sice disponují všemi funkcemi, které zdůrazňuji, ovšem jsou v rozporu s požadavky na jednoduchost, snadnou použitelnost, správu a nízkou cenu. Zbývá tedy Kontakt, který vyniká jednoduchostí a požadavky splňuje všechny – resp. téměř všechny. Nezmínil jsem, že navrhovaná aplikace bude určená pro platformu Windows – z důvodu, že podobně jednoduché nástroje se pro tuto platformu vyskytují jen zřídka. Ale o tom podrobněji až v další kapitole.

### 3 Specifikace

Kapitola shrnuje požadavky na výsledný produkt s ohledem na analýzu provedenou výše. Dostává se tak od základní koncepce až k elementárním funkcím programu, které by měly být součástí řešení.

#### 3.1 Celková koncepce aplikace

V kapitole 2.1 byly groupware systémy klasifikovány ze dvou hledisek – z hlediska architektury a z hlediska rozsáhlosti. V části 2.4.5 jsou charakterizovány vybrané aplikace založené na internetových technologiích. Již z úvodu této kapitoly je patrné, že internetových systémů je k dispozici velké množství, nabízí nejrůznější funkční vybavení, pro zákazníka tedy není těžké najít produkt, který vyhovuje jeho požadavkům. Z tohoto hlediska by navrhovaná aplikace nepřinesla nic nového, proto je zvolena architektura klient-server. To samozřejmě není jediným argumentem, proč nevytvářet webovou groupware aplikaci, další důvody vyplývají z obecné charakteristiky těchto aplikací v části 2.2. Jde především o uživatelský přívětivé grafické rozhraní a možnost s programem pracovat i bez připojení k internetu.

Nyní je nutné zvolit mezi CMT a CPMT softwarem. Velkých CPMT systémů sice na trhu mnoho není, ovšem tím obtížnější je nabídnout konkurenceschopné řešení v této oblasti – firmy IBM a Microsoft do svých groupware řešení investují každým rokem nemalé peníze. Má-li zákazník o takový software zájem, jen těžko nabídne nově vzniklý produkt podobné služby, včetně technické podpory, správy či školení uživatelů.

Z těchto důvodů byl zvolen koncept CMT. Ani takových aplikací není málo, nicméně tyto produkty se často diametrálně odlišují celkovým pojetím, případně nabízenými funkcemi. Abych to upřesnil, často jsem se setkával s tím, že funkce,

týkající se správy pracovních skupin, byly velmi omezené, nezdá se, že šlo jen o plánování úkolů do kalendáře – samozřejmě pro více uživatelů, jinak by nešlo o groupware. Jiným jevem, na který jsem při analýze dostupného software narážel, bylo, že nástroj pro správu pracovních skupin byl jen doplňkovou součástí programu, který byl primárně určen pro něco jiného. Šlo například o emailové klienty nebo programy pro správu dokumentů a jejich verzí. To samozřejmě nevylučuje možnost používat aplikaci výhradně jako CMT, ovšem proč pořizovat rozsáhlejší program s funkcemi, které nejsou potřebné. Výsledkem tedy je, že z CMT aplikací s tlustým klientem byl do analýzy zařazen pouze Contact, který dle mého názoru představuje vhodného zástupce CMT aplikace, a do značné míry jsem se jím nechal inspirovat. Nakonec je důležité zmínit, že produkt bude navrhován pro platformu Windows – na rozdíl od popisovaného Contactu.

Pominu-li v tuto chvíli základní funkce, které by měla aplikace obsahovat, je zde několik požadavků, které považuji za zásadní, a které do značné míry určují celkový ráz aplikace. V první řadě je podstatné, aby byla aplikace jasně vyhraněná, bylo na první pohled patrné, k čemu je určena a aby neobsahovala nadbytečné funkce, které nejsou nezbytné. Z pohledu uživatele se bude jednat o program, který mu bude zprostředkovávat informace o práci, kterou je potřeba udělat a zároveň mu poskytne zpětnou vazbu, díky které bude moci zjistit dodatečné informace, případně odevzdat některé výsledky své práce. Nic víc není nutné. Je to jedna z věcí, díky které se navrhovaný produkt bude odlišovat od ostatních. Vycházím z toho, že běžní uživatelé mají své zvyklosti. Používají e-mailové klienty, editory dokumentů či programy k online komunikaci, které jim vyhovují a z nějakého důvodu si je vybrali. Není v mých silách a ani nepovažuji za rozumné pokoušet se implementovat software, který by všemi takovými funkcemi disponoval, ve výsledku by jistě většina uživatelů zůstala u používání původních programů a aplikace by byla zbytečně složitá a těžkopádná. Naopak se domnívám, že je dobré v tomto ohledu vyjít uživatelům vstříc – navrhovaná aplikace bude disponovat mimo jiné adresářem a kalendářem, což jsou funkce vhodné k synchronizaci. Z toho důvodu by měly umožnit export dat v nějakém vhodném formátu, aby i tady dostal uživatel možnost sledovat události ve svém kalendáři, případně procházet kontakty v jiném poštovním klientovi. S touto problematikou souvisí také další požadavek, který bych chtěl zmínit. Jak již bylo řečeno, v případě, že se uživatel rozhodne i nadále používat pro některé funkce dosavadní programy, kterými disponuje, stanou se tyto funkce v programu nadbytečné, dělají jej nepřehledným a zbytečně jej zdržují. To je důvodem, proč by měla být aplikace plně modulární, aby bylo možné kteroukoli z částí programu vypnout a nepoužívat. Modularita má také několik dalších předností – software se stane snadno rozšiřitelným o další funkce a v případě, že bude dobře popsán rozhraní pro připojování modulů, budou se na rozšiřování programu moci podílet další vývojáři.

Jedním z méně výrazných požadavků na aplikaci je umožnění komunikace se serverem Kolab. Docílení stoprocentní kompatibility však možné nebude, neboť některé funkce, kterými bude aplikace disponovat, Kolab nepodporuje. Příkladem je sdílení souborů, respektive jejich asociace s projekty nebo rozdělení projektu na dílčí úkoly. Požadavek kompatibility tedy pro návrh aplikace bude

znamenat použití formátu dat a komunikačních protokolů, které jsou typické pro Kolab server. Detailní rozbor problematiky se nachází ve čtvrté kapitole.

Základní požadavky, které jsem tu stanovil, tedy shrnuji do několika vět: Cílem této práce je navrhnout klient-server aplikaci pro platformu Windows, která bude implementovat základní funkce nutné pro správu pracovních skupin a projektů. Klientská část programu bude modulární a bude brán zřetel zejména na přehlednost a intuitivnost grafického rozhraní. Díky požadavku modularity je možné další části kapitoly členit dle konkrétních modulů.

## **3.2 Definice a charakteristika částí aplikace**

Na základě požadavků uvedených v kapitole 2.5 lze poměrně přímočaře rozdělit funkční vybavení aplikace do jednotlivých komponent, které na sobě budou nezávislé a budou tedy fungovat odděleně. Jejich společným prvkem bude jednotné, dobře definované rozhraní, kterým budou komunikovat se zbytkem programu. Hlavní část aplikace bez jednotlivých modulů bude sloužit jako zdroj dat, který budou komponenty využívat. Nyní přistoupím k popisu konkrétních modulů, k výčtu funkcí a vlastností, které budou mít, a opět bych rád u každého modulu zdůraznil, co je pro něj zásadní, které funkce bude nutné implementovat a zcela jistě se objeví v konečné verzi aplikace, a co není nezbytné. O realizaci takových funkcí rozhodnou časové možnosti během vývoje aplikace. Nejprve ovšem uvádím obecnou charakteristiku projektů, k jejichž správě je vhodné program využít a v další podkapitole popisují základní skupiny uživatelů z hlediska oprávnění, které bude software rozlišovat.

### **3.2.1 Charakteristika projektů**

Rozdělování pracovních povinností, zadávání úkolů a celkový přehled řešených projektů, to jsou jen některé z elementárních vlastností, kterými by měl disponovat každý groupware. Přestože téměř každý úkol je možné dekomponovat na dílčí části, které je nutné postupně (případně paralelně) vypracovat, pokusím se nastínit, pro jaké typy projektů je program vhodný.

Vzhledem k tomu, že se software hodlá řadit do kategorie CMT groupware aplikací, využití pro projekty by mělo být dostatečně obecné. Jak jsem se již pokusil nastínit, v zásadě lze říci, že program bude využitelný pro jakékoli projekty, které je možné po dostatečně hluboké analýze rozdělit na dílčí úkoly, které jsou na sobě dostatečně nezávislé – to proto, aby mohly být přiděleny různým subjektům k vypracování. Jak by mohla vypadat obecná analýza projektu, jehož cílem je vybudování blíže nespécifikovaného internetového portálu, ukazuje následující tabulka. O závislostech by se dalo jistě polemizovat, stejně tak jako o dalším rozdělení úkolů. Například implementace či testování k tomu přímo vybízejí.

ID	Popis	Závislost	Zadáno	Vývoj	Termín
I.	Logický návrh	—	Jan Novák	50%	14. 12. 2009
II.	Design	—	Petra Tichá	100%	15. 11. 2009
III.	Implementace	I., II.	—	0%	31. 1. 2010
IV.	Testování	III.	—	0%	18. 2. 2010
V.	Doména + hosting	IV.	John Newman	0%	31. 1. 2009
VI.	Propagace	—	—	0%	—

**Tabulka 1 - Příklad projektu**

Vzhledem k tomu, že každý složitější projekt je předem analyzován a dělen na menší části, považuji za rozumné charakterizovat spíše ty, pro které vhodný není. Jsou to například takové projekty, které se neobejdou bez centrálně spravovaného úložiště provedených výsledků. Tím mám na mysli správu verzí či SVN<sup>3</sup>. S největší pravděpodobností tyto součásti nebudou implementovány v základní verzi aplikace, a proto je software primárně určen spíše pro středně velké a menší projekty, pro které není podstatné shromažďovat odvedenou práci na tomtéž místě a sdílet ji. Typickým příkladem projektů, pro jejichž zpracování není navrhovaný groupware zcela vhodný, jsou projekty softwarové. Při jejich vývoji není jednoduché pracovat bez zmíněné správy verzí, s jejíž implementací se v návrhu nepočítá.

### 3.2.2 Základní hierarchie uživatelských skupin

Jako v každé jiné aplikaci podobného zaměření, i tady je nutné udržovat určitou hierarchii uživatelů, resp. jejich oprávnění. Je-li tedy základní model takový, že uživatelé společnosti jsou rozděleni do pracovních skupin, skupinám jsou přidělovány projekty k řešení, projekty jsou děleny na dílčí úkoly a ty jsou teprve přiřazovány k realizaci konkrétním uživatelům, nabízí se přirozené členění uživatelů do následujících rolí:

- Správce pracovních skupin v rámci jedné společnosti, který může celé pracovní skupiny vytvářet, odstraňovat, určovat jejich vedoucí, vytvářet nové uživatelské účty pro zaměstnance společnosti a přiřazovat tyto uživatele do skupin.
- Vedoucí pracovní skupiny, který může v rámci skupiny, kterou administruje, vytvářet nové projekty, odstraňovat je, určovat vedoucí projektů a přiřazovat k projektům uživatele ze své pracovní skupiny.
- Vedoucí projektu, který může v rámci daného projektu vytvářet nové úkoly (má na starosti dekompozici projektu), přiřazovat tyto úkoly uživatelům, zařazeným do daného projektu a samozřejmě úkoly odstraňovat.
- Běžný uživatel, kterému program nebude sloužit pro správu projektů v pravém slova smyslu, ale poskytne mu přehled jeho pracovních povinností – jinými slovy úkolů, které dostal k vypracování. K tomu bude

<sup>3</sup> SVN – Subversion – systém pro správu verzí zdrojových kódů

mít k dispozici samozřejmě přehled dalších úkolů, na které je daný projekt rozdělen, případně přehled projektů, které řeší jeho pracovní skupina.

Program bude navržen tak, že bude možné v případě potřeby přidat další uživatelskou roli a přitom nebude nutné měnit implementaci aplikace, ovšem nepočítám s tím, že tato volba bude v aplikaci zpřístupněna.

Kromě uvedených rolí bude samozřejmě nutné, aby existoval správce, který bude moci přidávat celé společnosti a v rámci nich vytvářet účty správců pracovních skupin, aby bylo možné produkt zpřístupnit zákazníkům. Vzhledem k tomu, že se tato role nečlení k základním v oblasti správy pracovních skupin, nebude v návrhu programu existovat konkrétní definice uživatele s takovými právy, ale úkony, které jsem popsal, budou proveditelné pomocí příkazů na serverové části aplikace.

### **3.2.3 Správa projektů**

Správa projektů představuje ústřední komponentu, která bude uživateli zpřístupňovat největší a nejpodstatnější část dat. Díky tomuto modulu bude možné aplikaci nazvat groupware.

#### **3.2.3.1 Základní funkce modulu**

Komponenta by měla poskytnout rychlou a snadnou orientaci v projektech, na jejichž řešení se uživatel podílí. V úvodní části budou viditelné projekty, a to včetně detailů, jakými jsou například popis, důležitá data či poznámky. Následně bude možné vybrat si konkrétní projekt a dostat se k přehledu úkolů, ze kterých vybraný projekt sestává, opět včetně příslušných detailů.

Zobrazený obsah bude samozřejmě odpovídat tomu, jaké má uživatel oprávnění. Například vedoucímu pracovní skupiny budou přístupné veškeré projekty, které skupina vypracovává. To se týká i dalších funkcí nutných ke správě a organizaci takové aplikace. Mám na mysli vytváření nových projektů, úkolů či pracovních skupin, což jsou výsady privilegovaných skupin uživatelů.

V souvislosti se správou projektů je nutné zmínit funkci, kterou jsem ve shrnutí 2.5 označil jako rozšiřující pro obecnou groupware aplikaci, ovšem vzhledem ke koncepci tohoto programu ji považuji za poměrně zásadní. Jde o asociování souborů s jednotlivými projekty a úkoly. Textový popis projektu často není ideální k dostatečně konkrétní formulaci zadání, ovšem libovolný, rozumně velký přiložený soubor poskytne prostor pro data libovolného formátu (grafika, zdrojový kód apod.). Sdílení souborů u jednotlivých úkolů má stejnou výhodu a poskytuje také prostor pro zpětnou vazbu, neboť uživatel, řešící daný úkol, bude mít možnost přikládat k němu soubory, například části svého řešení, pokud k tomu bude úkol vhodný.

### 3.2.3.2 Další rozšíření správy projektů

Možností, jak vylepšit tento modul, se nabízí celá řada. Z těch, které s největší pravděpodobností budou ve finální verzi programu, jmenuji v první řadě poznámky – jednoduché rozšíření, které uživateli umožní dopsat ke každému projektu poznámku s libovolným textovým obsahem, stejné rozšíření se bude týkat dílčích úkolů, ovšem s tím rozdílem, že poznámku bude moci vložit pouze řešitel úkolu.

Vhodným doplňkem, který by mohl usnadnit komunikaci v rámci konkrétních skupin či projektů, je jakýsi lokální žurnál, komunikační kanál přidružený vždy ke konkrétnímu prvku (skupině, projektu). Do takového kanálu by mohli přispívat pouze uživatelé, kteří jsou členy odpovídající skupiny, resp. projektu. Lokální žurnály by tak uživateli poskytovaly rychlý přehled o tom, co se v konkrétní skupině děje a co je aktuální bez jiných příspěvků, které nejsou k tématu. Vzhledem k tomu, že jedním z dalších modulů bude žurnál na globální úrovni, neplánuji popsanou funkci implementovat.

Mnoho dalších užitečných nástrojů je založeno na shromažďování nejrůznějších dat, z analýzy provedené ve druhé kapitole jmenuji přehled času stráveného prací, logy, archivy realizovaných projektů a úkolů, ovšem žádná z uvedených funkcí není nezbytná a vzhledem k časovým možnostem s jejich implementací nepočítám. Stejně tak součástí aplikace nebudou další rozšíření, uvedená na konci druhé kapitoly. Důvody, proč nebudou implementovány, jsou uvedeny tamtéž.

### 3.2.4 Správa uživatelských skupin

Komponenta bude uspořádána podobně jako správa projektů. Jejím základním obsahem bude hierarchický přehled pracovních skupin a uživatelů, kteří do nich patří. Podobné schéma nabídne samozřejmě i pro projekty a jejich řešitele. Nabízí se několik možností, jak data o uživatelích uchovávat, například v relační databázi či adresářové struktuře LDAP. Charakteristika těchto řešení a rozhodnutí, která varianta byla vybrána, se nachází v návrhu aplikace.

#### 3.2.4.1 Základní funkce modulu

Pro běžného uživatele nebude tato část představovat nástroj pro správu, ale půjde o přehled toho, jak jsou uživatelé v dané pracovní skupině (případně skupinách) rozdělení. Díky tomu získá představu o svých spolupracovnících a o nadřízených. V přehledu bude samozřejmě zdůrazněno, kdo zastává jakou roli v dané pracovní skupině (resp. projektu).

Uživatel s vyššími právy využije komponentu k vytváření nových pracovních skupin, uživatelských účtů, ke stanovování vedoucích pracovních skupin a k rozdělování uživatelů do skupin a projektů. Na tomto místě bych rád uvedl, jaká jsou omezení, co se počtu privilegovaných uživatelů na skupinu týče. Správců pracovních skupin a vedoucích skupin může být libovolný počet, to považuji za rozumné. Jediná omezení, která se v návrhu nachází, jsou na počet vedoucích

projektů (pouze jeden uživatel) a počet řešitelů úkolu (také pouze jeden uživatel). Nemyslím si, že by to v důsledku představovalo značné omezení použitelnosti aplikace, a to z toho důvodu, že groupware slouží zejména k přidělování práce “na dálku” a i v případě, že má na stejném úkolu pracovat více lidí, nikdy by dva neměli dělat totéž. Nakonec doplním, že omezení se netýká počtu projektů, úkolů a skupin, ve které se konkrétní uživatel může nacházet, ten tedy může pracovat ve více skupinách najednou nezávisle na několika projektech.

#### **3.2.4.2 Další rozšíření**

V předešlém odstavci jsem zmiňoval omezení v počtu řešitelů jednoho úkolu, jedním z vylepšení tedy může být odstranění tohoto limitu. Jak jsem ovšem psal výše, nepovažuji to za důležité, ve výsledku je tedy třeba počítat s jedním uživatelem na jeden úkol. Přesněji s nejvýše jedním uživatelem, správci projektu bude umožněno nejdříve projekt dekomponovat a až pak jednotlivé úkoly přiřadit uživatelům.

### **3.2.5 Kontakty**

#### **3.2.5.1 Základní funkce**

Modul kontaktů bude pojat velmi jednoduše, půjde o běžný seznam osob, který nebude ničím výjimečný. Uživateli zobrazí ty osoby, které jsou zařazené ve stejných pracovních skupinách jako on. Samozřejmostí by mělo být zobrazení detailů u každého uživatele, vyhledávání a filtrování dle zvoleného kritéria (např. zobrazení pouze těch uživatelů, kteří přísluší k dané pracovní skupině). Aby bylo vyhověno požadavku nahraditelnosti modulu produktem třetí strany, je nutné implementovat export adresáře v nějakém vhodném formátu, například v XML souboru \*.contact, iCalendar nebo vCard. Tento formát bude ještě upřesněn.

#### **3.2.5.2 Rozšíření**

V popisu základních vlastností adresáře jsem úmyslně nezmínil možnost přidávat nové, vlastní kontakty. Touto funkcí adresář disponovat nebude, proto ji uvádím zde. V případě přidání takové funkce by bylo vhodné, aby mohl uživatel vytvářet své vlastní skupiny uživatelů nebo položky u kontaktů. Do budoucna by jistě bylo vhodné takovou možnost doplnit, nicméně pro základní použití groupware aplikace se mi uvedený adresář jeví jako dostačující. S tím souvisí import kontaktů, který také nebude obsažen ve finální verzi aplikace.

### **3.2.6 Kalendář**

Kalendář by měl především poskytnout přihlášenému uživateli přehled blížících se událostí, které se týkají projektů a úkolů, na kterých pracuje.

### **3.2.6.1 Základní vlastnosti**

Zobrazení počátečních a koncových termínů projektů a úkolů, na nichž se uživatel podílí. Informace by měly být zobrazeny přehledně a různé projekty barevně odlišeny. Vzhledem k charakteru událostí bude kalendář zobrazovat pouze měsíční přehled.

Synchronizace bude proveditelná exportováním a importováním událostí ve formátu iCalendar, který podporuje hojné množství produktů třetích stran (včetně MS Outlooku, IBM Lotus Notes a mnoha dalších jednoduchých kalendářů).

V kalendáři nebude omezeno vytváření nových událostí, jako je tomu ve správci kontaktů. Je nutné si uvědomit, že vedoucí projektu sice projekt rozdělí na dílčí úkoly, nicméně ani tyto části nejsou elementární, pouze určují, kterou část práce je povinen vykonat jeden uživatel. Proto považuji za vhodné nechat uživateli možnost vytvořit si vlastní jednoduché události, na které jej bude kalendář upozorňovat.

### **3.2.6.2 Možná rozšíření modulu**

Implementace propracovaného kalendáře by vystačila na vlastní aplikaci, možných vylepšení je opravdu mnoho. Z těch základních, o kterých by stálo za to uvažovat, jmenuji zobrazení týdenního přehledu (případně denního), klasifikace událostí dle různých typů či vylepšené možnosti synchronizace.

Za užitečnou považuji funkci popsanou u některých groupware produktů, a to synchronizaci kalendáře s jinými uživateli, případně sdílení kalendáře s uživateli z projektu a pracovní skupiny.

## **3.2.7 Žurnál**

Každý groupware by měl zprostředkovat alespoň minimální komunikaci mezi uživateli. Do návrhu aplikace jsem zvolil žurnál v podobě jednoduchého prostředku, který bude zobrazovat příspěvky uživatelů v rámci jedné pracovní skupiny. Dalšího rozlišení se příspěvky nedočkají – komentáře k jednotlivým projektům a úkolům je možné vkládat prostřednictvím poznámek, o kterých jsem psal v popisu modulu správy projektů. Žurnál by měl fungovat na globální úrovni.

### **3.2.7.1 Základní vlastnosti modulu**

Běžnému uživateli bude umožněno vkládat příspěvky do vláken, které budou tvořeny pracovními skupinami, ve kterých je zařazen. Příspěvky budou textového charakteru.

### **3.2.7.2 Rozšiřující funkce**

Žurnál je obecný název zahrnující celou řadu způsobů komunikace, proto je možné zvolit několik odlišných způsobů, jak komponentu dál rozvíjet. Přikláněl bych

se k vytvoření diskusního fóra, kde by i běžní uživatelé mohli vytvářet vlákna a diskutovat v nich. Jistě by se komunikace stala přehlednější a strukturovanější.

### **3.3 Závěrečné srovnání funkcí**

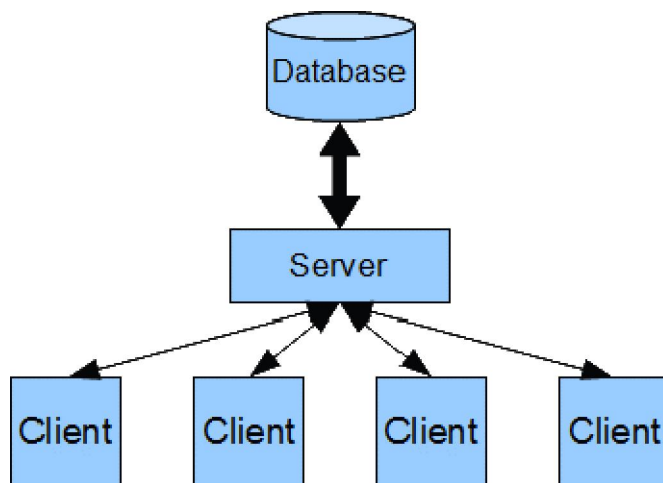
Nyní bych rád shrnul získané poznatky o groupware produktech a jejich funkcích. Uvedená tabulka srovnává groupware aplikace, probrané ve druhé kapitole a v posledním sloupci s těmito produkty porovnává produkt, navrhovaný v této kapitole. Poslední sloupec tabulky je vyplněn s ohledem na základní funkce, které byly u každého modulu popsány. Je patrné, že navrhovaná aplikace je funkčním vybavením velmi podobná PHP Projectu. Jediným parametrem, který PHP Project ve srovnání s aplikací postrádá, je přístup k datům pomocí tlustého klienta. To je však natolik významný rys, že aplikace jsou diametrálně odlišné. PHP Project je zástupcem groupware implementovaného pomocí internetových technologií, který uživateli nezprostředkuje přístup k datům bez připojení k internetu. Naproti tomu navrhovaná aplikace tlustým klientem disponuje a je schopná uchovat data offline.

Aplikace	IBM Lotus Notes	MS Exchange Server	Kontakt	Dot Project	More group ware	PHP Project	Návrh aplikace
Kontakty	+	+	+	+	+	+	+
Kalendář	+	+	+	+	+	+	+
Správa projektů	+	+	+	+	+	+	+
Úkoly	+	+	+	+	+	+	+
Fórum (komunikace)	+	+	+	+	+	+	+
Přehled	+	—	+	—	+	+	+
E-mail	+	+	+	—	+	+	—
Sdílení souborů	—	+	—	—	—	+	+
Přístup přes www	+	+	—	+	+	+	—
Správa uživatelů	+	+	+	+	+	+	+
Uživatelské skupiny	+	+	+	—	+	+	+
Poznámky	+	+	+	—	+	+	+
Web browsing	+	—	—	—	—	—	—
Open source	—	—	+ (GPL)	+	— (free)	+	+
Modularita	částečně	+	+	+	—	+	+
Podpora více serverů	+	+	+	—	—	—	—
Tlustý klient	+	+	+	—	—	—	+

Tabulka 2 - Srovnání analyzovaných groupware aplikací

## 4 Návrh aplikace

Tato kapitola se věnuje konkrétnímu návrhu aplikace, který splňuje požadavky uvedené v kapitole předešlé. Kromě nároků na funkční vybavení, uváděných postupně u popisu jednotlivých komponent, bych zdůraznil charakteristiku celkové koncepce, která je pro návrh velmi podstatná a určuje některé významné rysy aplikace. Tou je rozdělení na serverovou a klientskou část. Stejným způsobem je dělena i tato kapitola. Součástí kapitoly je také návrh databáze, který je řazen na konci. Pro ilustraci je přiloženo schéma typické aplikace architektury klient-server s datovým úložištěm.



Obrázek 1 - Schéma klient-server aplikace s databází

## 4.1 Problematika kompatibility se serverem Kolab

Ve specifikaci byl zmíněn požadavek na kompatibilitu s Kolab serverem. Zároveň bylo vysvětleno, proč tomuto požadavku nelze zcela vyhovět – znamenalo by to omezení některých funkcí, které specifikace pokládá za nepostradatelné. Tato kapitola vysvětluje, jak Kolab server funguje, jaké využívá technologie a určuje, v čem konkrétně bude aplikace s Kolabem kompatibilní.

Navrhovaný software sestává ze tří vrstev: Klientské části, aplikačního a databázového serveru. Server Kolab slouží k uchovávání dat a k jejich distribuci klientům. To jsou služby, které v navrhovaném programu přísluší aplikačnímu serveru a databázi. Pokud by tedy bylo docíleno úplné kompatibility s Kolabem, bylo by možné tyto dvě části aplikace nahradit právě jím. Cílem práce ovšem není program, který by nabídkou funkcí odpovídal Kolabu. Z toho důvodu aplikační server s databází nabízí odlišné možnosti využití. Jde o zmíněné sdílení souborů a další funkce, popsané ve specifikaci. Koncept Kolabu je však velmi dobrý, rozšířený a je přínosné zohlednit v návrhu způsob, jakým Kolab funguje. Jde o komunikační protokoly a formát přenášených dat. Díky tomu bude v budoucnu snadné omezit některé funkce klientské části a umožnit, aby místo nativního serveru komunikovala s Kolabem.

Základními entitami, se kterými umožňuje Kolab pracovat, jsou události v kalendáři, povinnosti, poznámky a kontakty. Tato data jsou hierarchicky ukládána do adresářové struktury do IMAP složek ve formě XML souborů, přičemž IMAP<sup>4</sup> server zajišťuje správu přístupových práv k těmto datům. Adresářová služba LDAP slouží k uchovávání informací o uživateli a nastavení serveru. Přenos dat je realizován protokoly IMAP a SMTP<sup>5</sup>. Kolab se snaží využívat otevřené formáty, proto

<sup>4</sup> *Internet Message Access Protocol* – protokol pro vzdálený přístup k e-mailové schránce

<sup>5</sup> *Simple Mail Transfer Protocol* – protokol zajišťující doručení pošty pomocí přímého spojení mezi odesílatelem a adresátem, zpráva je tak k dispozici offline

jsou data na aplikační úrovni přenášena zprávami ve formátu MIME<sup>6</sup>. Jde o stejný formát, který slouží k přenosu elektronické pošty. Vlastní data, tedy jednotlivé entity, jsou distribuovány jako přílohy zpráv MIME. Opět platí, že jsou použité otevřené formáty. Pro kontakty je zvolen formát vCard, pro události v kalendáři iCalendar, oba dva otevřené a standardizované IMC<sup>7</sup>. Pokud tedy klient požaduje data, připojí se k serveru, procházením IMAP složek najde XML soubor, který chce stáhnout, obsah tohoto souboru zkonvertuje do formátu vCard nebo iCalendar, vytvoří novou MIME zprávu a zkonvertovaný soubor do zprávy vloží jako přílohu. Kolab využívá také další komunikační protokoly, jako jsou POP3, FTP a HTTP, ovšem ty zprostředkovávají funkce, které nejsou pro tento návrh podstatné.

V aplikaci budou pro přenos dat použity zmíněné formáty vCard a iCalendar, které jsou standardizované a vyhoví tak nejen požadavku kompatibility, ale také synchronizace s jinými aplikacemi. Nehledě na to, že oba dva formáty již byly ve specifikaci zmíněny bez ohledu na kompatibilitu s Kolabem. Využit bude také formát zpráv MIME, který umožní snadný přenos dat různého typu a využitelnost přenášených zpráv nejen ke komunikaci se serverem Kolab, ale také k výměně informací s jinými aplikacemi.

Důležitým rysem serveru Kolab je způsob, kterým uchovává konfigurační data a informace o uživatelských účtech. Jde o adresářovou službu LDAP. Není to ovšem jediná možnost, jak data podobného charakteru ukládat. LDAP je optimalizována pro rychlé vyhledávání dle různých kritérií. Data jsou uspořádána hierarchicky. Problémem této struktury je pomalejší zápis a mnoho operací, které je nutné vykonat při reorganizaci dat nebo při změně jejich struktury. Další nevýhodou je náročná implementace. Jinou možností je uchování dat v relační databázi, která vyniká rychlostí poskytování dat a jednoduchostí nasazení. Data v ní nejsou uspořádána hierarchicky. Nevýhodou je, že použití relační databáze je podmíněno využitím samostatného databázového serveru. Dojde tak k rozšíření architektury o další vrstvu. Vystává tedy otázka, jaký koncept zvolit pro navrhovanou aplikaci. Pro LDAP hovoří fakt, že jej využívá Kolab. Proti je kromě implementační náročnosti a rychlosti také méně přehledná a uživatelsky přívětivá správa struktury a větší nároky na hardwarové vybavení serveru. Z těchto důvodů návrh počítá s vytvořením aplikace o třech vrstvách, přičemž data budou uchována v relační databázi. Aby byla zachována možnost použití serveru Kolab, je nutné odpovídajícím způsobem navrhnout komunikační rozhraní, protokoly a formát přenášených dat.

## 4.2 Serverová část

Serverová část aplikace představuje pro klienty prostředníka pro komunikaci s databází. Aplikace má tedy tři vrstvy: Klientskou část, serverovou část a databázový server, přičemž s databází komunikuje část serverová. Nejprve

---

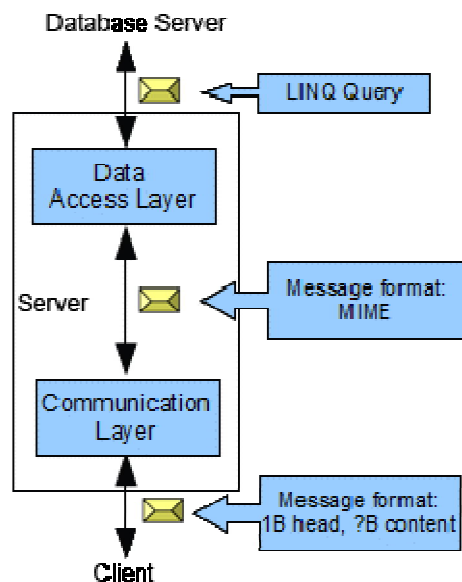
<sup>6</sup> Multipurpose Internet Media Extensions – standard, umožňující do těla a hlavičky zpráv elektronické pošty vkládat text s diakritikou, přikládat ke zprávám přílohy a dělit zprávy na více částí

<sup>7</sup> Internet Mail Consortium – sdružení zabývající se standardy elektronické pošty

realizuje autentizaci a pak na základě zpráv, které si s klienty vyměňuje, zprostředkovává klientům data z databáze a soubory uchovávané na serveru. Výměna dat je obousměrná, tedy server také přijímá nová data a soubory od klientů a patřičně s nimi nakládá, nicméně lze očekávat, že tok dat ve směru od serveru bude mnohonásobně větší a je potřeba tomu návrh přizpůsobit. Dalším důležitým faktem je, že server bude vždy pouze jeden, alespoň tak s tím počítá tento návrh, proto je nutné brát ohled na to, aby nebyl zatěžován zbytečnými operacemi, které jsou realizovatelné na straně klienta.

## 4.2.1 Dekompozice návrhu serveru

Návrh serverové části aplikace je strukturovaný na základě funkcí, které implementuje a rozdělený do vrstev, protože je třeba zajistit komunikaci na různých úrovních. Každá následující podkapitola odpovídá jedné takové vrstvě. Z uvedených požadavků vyplývá, že je nutné implementovat dvě odlišné a samostatně dostatečně významné části programu. První z nich obslouží komunikaci s klienty, druhá zajistí přístup do databáze a vše ostatní s ní spojené. Pro ilustraci je přiloženo schéma. Přesný tvar zpráv, které si vrstvy vyměňují, je popsán v následujících kapitolách.



Obrázek 2 - Vrstvy serverové části aplikace a komunikace mezi nimi

### 4.2.1.1 Komunikace s klientem

Tato vrstva zajišťuje komunikaci mezi serverem a klienty. Zprostředkovává pouze fyzické zasílání a přijímání zpráv, přičemž zprávy rozlišuje na

- **servisní** – slouží k realizaci spojení, informaci o jeho stavu a k jeho ukončení

- **komunikační** – předávané beze změn vrstvě nadřazené.

Stejným způsobem zachází se soubory, jejichž přenos zajišťuje také oběma směry.

Komunikace probíhá asynchronně, tedy bez nuceného čekání, každý klient je obsluhován vlastním vláknem, které se vytvoří po klientově připojení a reprezentován instancí třídy, která o něm shromažďuje informace. Spojení je realizováno pomocí socketů, zprávy tedy není nutné adresovat, server musí při posílání pouze zvolit konkrétní socket, na kterém klient naslouchá. Vrstva je poměrně důsledně oddělena od vrstev ostatních, nemá tedy žádný přístup k datům z databáze a nemá ani informaci o tom, zda je připojený klient autentizován či nikoliv.

Zprávy komunikační vrstvy, pomocí kterých se dorozumívají klient se serverem, mají velmi jednoduchý formát, k rozlišení typu zprávy slouží první byte, zbytek tvoří obsah. Typy zpráv jsou popsány v tabulce.

Zápis rozlišovacího bytu v desítkové soustavě	Význam zprávy
98	Servisní zpráva, která informuje druhou stranu o tom, že je možné začít s posíláním proudu dat.
100	Servisní zpráva, informující server o klientově odpojení.
107	Servisní zpráva, kterou zasílá server klientovi v případě, že se podařilo schválit žádost o stažení souboru.
109	Zpráva vyšší komunikační vrstvy. Taková zpráva je rozbalena (zbavena hlavičky) a předána vrstvě vyšší.
110	Servisní zpráva, informující server o připojení nového klienta.
111	Servisní zpráva, sloužící především k testování spojení a potvrzování.
119	Servisní zpráva, kterou informuje server klienta o úspěšném navázání spojení.

**Tabulka 3 - zprávy komunikační vrstvy**

#### 4.2.1.2 Přístup k datům z databáze

Druhá vrstva vyřizuje žádosti ze strany klienta na poskytnutí dat. Převádí formulace zpráv do dotazovacího jazyka a tyto dotazy klade datovému zdroji. Následně zpracovává výsledky do přijatelné formy, serializuje je, aby bylo možné je odeslat, a vytvoří z nich zprávu ve formátu MIME, kterou předává jako data k odeslání nižší vrstvě. Je tedy jisté, že právě operace na této vrstvě představují pro server největší zátěž během celého procesu obsluhy klienta.

Pro implementaci databázových dotazů byl zvolen integrovaný dotazovací jazyk LINQ, který poskytuje objektový pohled na data a s tím související typově jasně definované prostředí. Výsledkem je rozhraní pro přístup do SQL databáze, přičemž dotazy jazyka LINQ jsou mapovány na patřičné SQL dotazy s tím efektem, že jsou formulovány v nativním programovacím jazyce. Jinými alternativami jazyka LINQ, které nebyly vybrány, ale přicházejí v úvahu, jsou Entity Framework nebo prosté formulování dotazů pomocí textových řetězců, psaných do zdrojového kódu. Druhá

varianta byla vyloučena, neboť nenabízí nic z toho, co jazyk LINQ. Zbývá Entity Framework, což je, podobně jako LINQ, struktura, sloužící k mapování relačních dat na objektová. Jde o novější technologii, jejíž hlavní předností je možnost měnit mapování na fyzický model, aniž by musel být změněn model konceptuální – tedy ten, kterým byla vytvořena struktura, jež je v programu použita. Mapování je tedy v jistém smyslu volnější. Jazyk LINQ byl upřednostněn z toho důvodu, že jde o jednoduchou technologii s přímočarým mapováním. Oproti Entity Framework představují dotazy jazyka LINQ menší nároky pro SQL server. To je dáno právě těsností mapování, kterým je LINQ omezen. Nebude-li tedy využít potenciál Entity Framework, je zbytečné nasazovat robustnější technologii.

Veškeré žádosti, které vyřizuje server, jsou inicializovány klientem. Je možné je rozdělit následujícím způsobem:

- Žádosti o zaslání dat z databáze. Jde o typické požadavky klienta na potřebná data, která vedou k rozsáhlým databázovým dotazům a následně k odesílání takto získaných dat pomocí zpráv MIME na stranu klienta. Typickým příkladem je poskytnutí projektů, na kterých se přihlášený uživatel podílí, nebo informací o uživateli, se kterými spolupracuje.
- Požadavek na vložení nových dat do databáze. Méně často uživatel potřebuje do databáze data vkládat, proto se předpokládá, že proudy takových dat budou podstatně menší. Pokud je vše v pořádku, data jsou vložena do databáze. Vyskytne-li se chyba, klient je o problému informován zasláním zvláštní zprávy s informací o tom, že vložení dat neproběhlo.
- Požadavek na aktualizaci existujících dat. Jde o žádost, která je vyřizována stejně, jako při vkládání dat nových.
- Žádost o přenos souboru směrem ke klientovi. Uživatel se dožaduje stažení souboru, server stahování tedy zahájí a počká, potvrdí-li klient úspěšné přijetí souboru.
- Žádost o přenos souboru na server. Jde o situaci, kdy se uživatel rozhodne uploadovat na server nový soubor. Pro takové situace je zvolen limit velikosti souboru na 30MB. S ohledem na charakter souborů, které se budou k projektům přikládat, to považuji za dostatečnou velikost. O velikosti limitu je samozřejmě možné polemizovat, případně jej zvýšit. Nebyl stanoven na základě žádného fyzického ani implementačního omezení a bude možné jej jednoduše měnit.
- Jiný typ žádosti. Většinou jde o požadavky, které vyžadují primitivní dotazy do databáze a jednoduché textové odpovědi. Typickým příkladem je autentizace či odhlášení uživatele.

Zprávy, které předává komunikační vrstva přístupové, dodržují formát definovaný standardem MIME. Důvodem pro použití tohoto tvaru zpráv je zachování kompatibility se serverem Kolab, který realizuje přenos dat stejným způsobem. MIME zprávy, které používá, odpovídají doporučení RFC 2445. Zpráva

v tomto formátu tvoří obálku, vlastní data jsou do zprávy vkládána jako přílohy textového charakteru. Přesný tvar těchto dat je popsán v následující kapitole. Zpráva ve formátu MIME sestává z následujících položek:

- `MIME-version: 1.0` značí, že jde o zprávu ve formátu MIME.
- `Content-type:` označuje typ obsahu v těle zprávy. V této implementaci bude položka nabývat následujících hodnot: `multipart/mixed` pro označení vícedílné zprávy. Za tímto údajem musí následovat definice oddělovače jednotlivých částí, například `boundary="frontier"`. Další hodnotou pro typ obsahu je `text/plain` pro přenos neformátovaného textu, `text/calendar` pro přenos dat ve formátu iCalendar, `text/x-note` pro přenos poznámky a `text/x-vcard` pro přenos kontaktu ve formátu vCard. Poslední tři uvedené typy dat odpovídají specifikaci zpráv serveru Kolab.
- `Content-transfer-encoding:` určuje kódování použité pro obsah zprávy.
- Součástí jsou i další položky, například `From`, `To` nebo `Subject`, ty však pro implementaci nejsou podstatné.

Aby bylo možné rychle se zprávami pracovat, vytvářet je a číst jejich obsah, bude nutné implementovat knihovnu, která tyto činnosti obstará. Využití tato knihovna najde v serverové i klientské části programu.

## 4.2.2 Formát dat přenášených mezi serverem a klientem

Z výsledků databázových dotazů se stávají objekty, které jsou serializovány, baleny do MIME zpráv a zasílány na stranu klienta. Aby u klienta mohla být data bez potíží přečtena, je nutné, aby měla stejný formát. Pro ukládání dat získaných z databáze bude využit formát, který definuje zvláštní knihovna, sdílená serverem i klientem – aby bylo možné data jednoduše serializovat a deserializovat na obou stranách. Tento formát musí být dobře navržený, neboť, jak bude později popsáno, také klient bude muset držet databázová data ve svých strukturách. Budou-li tedy struktury navrženy vhodně, budou využitelné i u klienta. V kapitole 4.1, kde jsou charakterizovány vybrané standardy, kterými se řídí server Kolab, byly uvedeny formáty dat, které Kolab využívá. Jde o formáty otevřené a hojně užívané. Nejčastěji mají podobu značek, definujících jednotlivé položky objektu a jejich hodnotu.

Formát vCard slouží k přenosu informací o kontaktech a uživatelích, je implementován s ohledem na RFC-2425 a obsahuje tyto značky:

- `BEGIN:VCARD`, `END:VCARD` k označení začátku a konce jednoho objektu.
- `VERSION:` k určení verze vCard
- `N:` označuje celé jméno ve tvaru příjmení;jméno
- `FN:` označuje celé jméno ve tvaru, v jakém se běžně píše (jméno příjmení)

- ORG: nese jméno organizace asociované s danou vCard
- ADR: označuje adresu
- TEL;TYPE=WORK;VOICE: uchovává telefonní číslo do práce
- EMAIL: nese informaci o e-mailovém kontaktu
- Formát vCard umožňuje vložení dalších položek, které mohou sloužit jako rozšiřující. Takové položky musí začínat znaky X-, tedy například X-NICK, kterou využívá navrhovaná aplikace k uložení přihlašovacího jména osoby.

Formát iCalendar je velmi podobný. Jde o standard pro výměnu kalendářových dat (událostí, úkolů a položek žurnálu), popsáný v dokumentu RFC 5545. V aplikaci je tento formát využit k přenosu informací o projektech a úkolech stejně, jako je tomu u serveru Kolab. Sestává z následujících položek:

- BEGIN:VCALENDAR, END:VCALENDAR uvozují a ukončují soubor událostí, které kalendář obsahuje.
- BEGIN:VEVENT, END:VEVENT obklopují konkrétní událost.
- DTSTART:, DTEND: určují datum počátku a konce události. Datum je zapsáno ve tvaru 19993101T23:59:59Z, v tomto případě jde o 31.01.1999 23:59:59.
- SUMMARY: slouží k popisu události
- Opět je možné použít vlastní deskriptory začínající dvojicí znaků X-.

## 4.3 Klientská část

Návrh klienta je podstatně složitější, neboť je potřeba zohlednit celou řadu výše definovaných požadavků. Mezi ty nejdůležitější patří modularita a schopnost poskytovat data i bez přístupu k internetu, tedy k serverové části. Přirozeně je nutné implementovat stejný komunikační protokol, který byl použitý u serveru. Při dekompozici budu probírat postupně jednotlivé požadavky a určovat, jakým způsobem ovlivní návrh aplikace, případně jaké třídy a vrstvy bude nutné do návrhu zařadit. Na konci se pokusím tento rozbor shrnout a vyvodit z něj konečnou podobu klientské části aplikace.

### 4.3.1 Dekompozice návrhu

Prvním diskutovaným požadavkem je modularita. Aplikace se proto dělí na pluginy (resp. moduly) a hlavní část, která obsahuje jádro a hlavní formulář, který poskytuje prostor jednotlivým modulům. Jádro samo o sobě zprostředkovává komunikaci mezi moduly a hlavním formulářem, případně pouze mezi jednotlivými pluginy, je-li to nutné. Aby bylo jasné, jak přesně je modularita zohledněna v návrhu programu, budou zde jednotlivé komponenty a rozhraní, které slouží ke komunikaci mezi pluginy, blíže popsány. Po každém modulu je vyžadováno, aby definované

rozhraní implementoval a tím se stal připojitelným k hlavnímu programu. Stejně tak jádro aplikace implementuje rozhraní, které poskytuje všem modulům a definuje, co mohou pluginy od aplikace očekávat a s čím počítat. Díky tomu je implementace jádra nezávislá na pluginech, je možné ji snadno aktualizovat a tvorby nových modulů se mohou ujmout programátoři s implementací jádra neobeznámení.

Moduly v této aplikaci představují hlavně prostředky pro zobrazování a práci s daty. Liší se od sebe zejména typem dat, se kterým pracují. Hlavním cílem každého z modulů je především poskytnout uživateli příjemné a jednoduché prostředí pro práci s konkrétními údaji. S ohledem na kapitolu 3.1 je jednou z hlavních výhod maximální možná nezávislost modulů a možnost kterýkoli z nich vypnout. Přímá komunikace mezi pluginy proto neprobíhá vůbec nebo je velmi omezená.

Dalším tématem je poskytování dat. Nejprve je potřeba určit, do jaké části aplikace je vhodné zařadit tuto vrstvu. Nejméně vhodným řešením by bylo implementovat poskytovatele v každém modulu zvlášť s ohledem na data, se kterými modul pracuje. I přes to, že se pluginy odlišují právě těmito daty, rozhodně nepracují s disjunktními částmi dat. Například modul pro práci s projekty potřebuje znát informace o uživatelích, kteří se na projektech podílejí a vedou je, čímž zasahuje do domény dat pluginu pracujícího s organizací uživatelů do skupin. Daleko lepším řešením je začlenit vrstvu do hlavní části programu a v rozhraní pro komunikaci s klienty popsat metody, jak k datům konkrétního typu přistupovat.

Vrstva poskytování dat by měla dostát i požadavku na přístup k datům v offline režimu. Tedy měla by modulům poskytovat ta nejaktuálnější možná data, která má k dispozici. V případě, že je k dispozici připojení k serveru, zprostředkuje zaslání dat ze serveru. Pokud není možné se připojit, poskytne modulu neaktuální data ze svého offline úložiště v paměti. Také se musí postarat o to, aby po ukončení práce s programem byla data stažená ze serveru uložena na pevný disk. V případě, že při dalším spuštění nebude k dispozici připojení k serveru, načtou se data z disku. Tato vrstva bude rovněž zprostředkovávat vkládání nových dat do databáze. Pokud jsou na server odeslána a serverem uložena nová data, klientská aplikace vloží tato data do svých existujících struktur a nebude je muset nahrazovat novými až při následném čtení. Dojde tak ke zmenšení potřeby komunikovat po síti.

Zapouzdření offline a online poskytování dat neznámá, že pro moduly bude informace o existenci (resp. neexistenci) spojení na server nedostupná. V režimu offline není možné vkládat nová data. V opačném případě by se databáze mohla dostat do nekonzistentního stavu.

Dále je třeba přizpůsobit se komunikaci probíhající se serverovou částí aplikace. Vrstva, která na serveru komunikuje s databází, odesílá klientovi zprávy ve formátu MIME. Je tedy nutné zařadit do návrhu klienta odpovídající vrstvu, která bude umět se zprávami pracovat. Nakonec je nutné implementovat komunikační vrstvu, která realizuje přenos servisních a komunikačních zpráv stejně, jako je tomu u serveru.

### **4.3.2 Rozhraní pro komunikaci s jádrem**

V této sekci je popsáno rozhraní, které implementuje jádro programu, a které pro moduly představuje vstupní bod, jež mohou využívat. Základem jsou data, která jádro poskytuje. Jelikož každý modul vyžaduje zpřístupnění trochu odlišných dat, jsou metody rozděleny dle typu dat, která může modul požadovat. Jde o pracovní skupiny, projekty, uživatele, položky žurnálu či kalendářové události. Poskytnuta budou samozřejmě vždy jen relevantní data, ke kterým má přihlášený uživatel přístup, viz přístupová vrstva v serverové části aplikace.

Dále jsou modulům poskytnuty metody pro získání doplňkových dat, která není nutné zasílat společně s daty základními. Jde například o soubory, které jsou přiřazeny k projektům, poznámky atp. Jedná se o informace, ke kterým není přistupováno příliš často a může se stát, že k jejich vyžádání vůbec nedojde.

Rozhraní musí umožnit rovněž vkládání dat nových, a to pracovních skupin, projektů, uživatelů a úkolů a jejich vzájemných vazeb.

Poslední funkcí, kterou rozhraní zprostředkuje modulům, je odesílání a přijímání souborů.

### **4.3.3 Rozhraní pro komunikaci s moduly**

Toto rozhraní je podstatně jednodušší, neboť jádro většinu času žádnou komunikaci s pluginy neinicuje. Stačí tedy definovat vlastnosti nutné k identifikaci modulu (jméno, případně popis, verze), dále metody volané při inicializaci modulu a samozřejmě vlastnost, která definuje prvek uživatelského rozhraní, který bude zobrazen v hlavní části programu.

### **4.3.4 Vrstva poskytování dat**

O této vrstvě toho bylo mnoho řečeno již v dekompozici návrhu v kapitole 4.2.1. Vrstva si udržuje data získaná ze serveru ve vlastních strukturách a v případě potřeby je aktualizuje. Pokud přestane být k dispozici spojení se serverem, poskytuje data ze svých struktur s informací, že nemusejí být aktuální a odmítá pokusy o vkládání nových dat do databáze. Při ukončení programu uloží data získaná ze serveru na pevný disk, aby bylo možné k nim přistoupit v případě, že při dalším spuštění programu nebude připojení k serveru k dispozici.

### **4.3.5 Vrstva přístupu k datům**

Název vrstvy může být poněkud zavádějící. Jde o alternativu vrstvy přístupu k datům, která je implementována na serveru. Tato vrstva ovšem nepřistupuje k datům z databáze, ale k datům, která poskytuje aplikační server. Opět je brán ohled na kompatibilitu se serverem Kolab. Jde o vrstvu na druhé nejnížší úrovni, přičemž tvar zpráv odpovídá příkazům protokolu IMAP. Jde o jednoduché textové příkazy, sloužící k přihlášení na server a vyžádání dat. Příchozí data ze serveru mají formát zpráv MIME, stejně tak jako nová data, která mají být na server uložena. Vrstva je navržena tímto způsobem, neboť server Kolab se při poskytování dat

chová jako IMAP server. Kolab uchovává data ve složkách v souborech XML, které při žádosti o data konvertuje do popsaných formátů vCard nebo iCalendar a balí je do MIME zpráv. Aplikační server navrhovaného programu se navenek chová velmi podobně. Data poskytuje stejným způsobem, ovšem místo XML souborů je získává z relační databáze. Díky stejnému komunikačnímu rozhraní zůstává klientovi tento rozdíl skrytý a je schopný výměny dat s Kolabem i aplikačním serverem.

Zprávy této vrstvy jsou předávány vrstvě nižší, která realizuje fyzické spojení se serverem. Výsledkem je, že nadřazená vrstva nemusí formulovat přesný tvar zpráv pro získání dat určitého typu, což by nebylo vhodné. Stejným způsobem by rozhraní muselo formulovat zprávu pro vyžádání stažení souboru, což je dokonce zpráva nejnižší komunikační vrstvy.

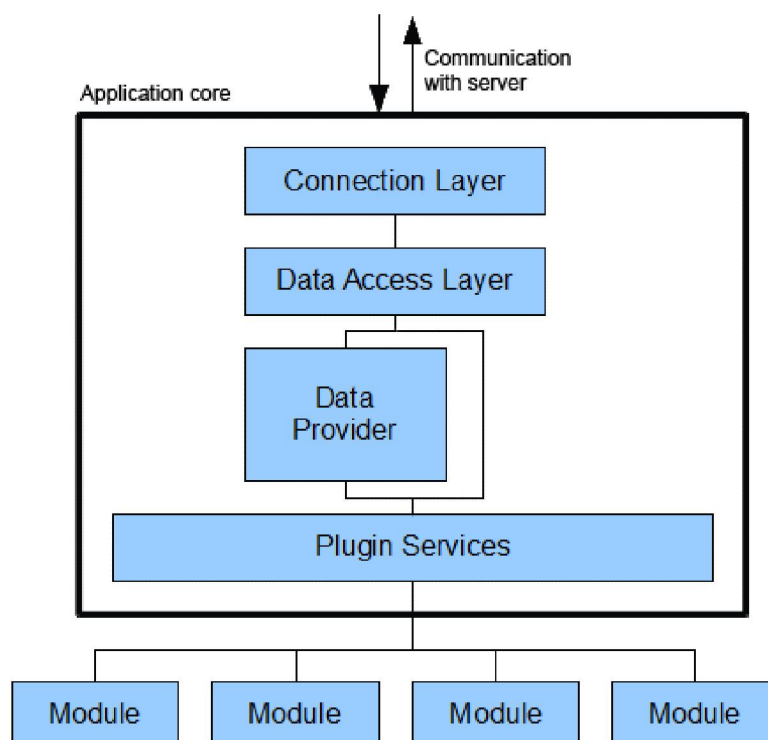
### **4.3.6 Komunikace se serverem**

Komunikace je v návrhu programu realizována vrstvou na nejnižší úrovni, která je velmi podobná odpovídající vrstvě na serveru. Jde tedy o asynchronní komunikaci implementovanou pomocí socketů, soubory jsou přenášeny pomocí streamů a komunikační zprávy mají stejný formát a význam jako u serveru, nebudu je zde proto znovu uvádět. Obdobně jako server balí zprávy vyšší vrstvy do vlastního formátu a zasílá je serveru jako komunikační, případně pomocí servisních zpráv iniciuje zasílání streamů či vytváření spojení.

### **4.3.7 Shrnutí**

Výsledná struktura programu je poměrně složitá, na tomto místě je vhodné dodat, že nebyly popsány všechny vrstvy, které se v programu vyskytují. Mohlo by se například zdát, že rozhraní, které využívají moduly pro komunikaci s jádrem, bude implementováno poskytovatelem dat. Ve skutečnosti existuje mezi tímto rozhraním a poskytovatelem ještě jedna vrstva, která rozhraní implementuje. Nese příznačné označení služby pluginům. Prvním důvodem pro její existenci je, aby poskytovatel dat mohl být implementován dostatečně nezávisle. Druhým argumentem je fakt, že rozhraní neobsahuje pouze metody pro přístup k datům, ale například metody pro vyžádání stažení souboru, což jsou operace, které nemusí poskytovatel dat řešit. Není vyloučeno, že takových metod bude přibývat a rozhraní bude rozšířeno. Na rozdíl od změny rozhraní pro pluginy by byla zachována kompatibilita modulů staršího data.

Pro lepší představu o celkové podobě aplikace je přiloženo ilustrační schéma.



Obrázek 3 - Návrh vrstev klientské části aplikace

## 4.4 Návrh databáze

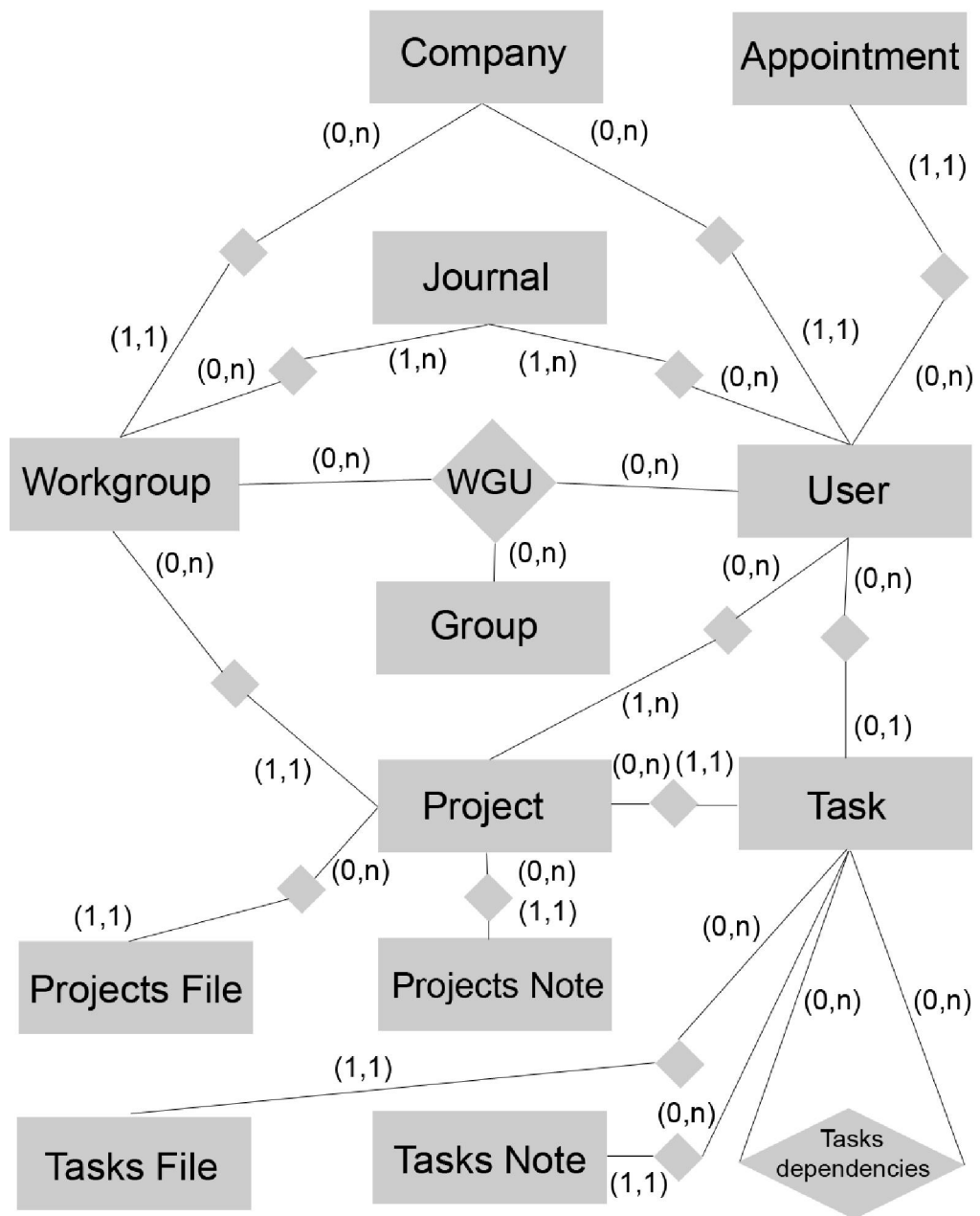
Databáze je centrálním úložištěm dat, klient a server slouží jen ke zprostředkování těchto dat uživateli, proto je návrhu databáze věnována patřičná pozornost. Ve třetí kapitole byly popsány funkce, kterými bude program disponovat. Je evidentní, že má-li být všem požadavkům vyhověno, musí být databáze poměrně rozsáhlá.

Žádná z předchozích kapitol nerozebírala, jak zajistit, aby byla do databáze vkládána korektní data a databáze zůstala v konzistentním stavu. Bylo by možné provádět kontrolu vstupních dat na serveru, případně u klienta, ovšem takové řešení by představovalo zbytečné náklady na transport dat, která jsou jinak nevyužitelná. Nehledě na to, že by server byl zbytečně zpomalován operacemi, které by na starosti mít neměl. Rychlejší a vhodnější řešení je implementace databázových triggerů a integritních omezení, které se o kontrolu dat postarají. Server tak nadále zůstane zprostředkovatelem mezi klientem a databází, ať půjde o potvrzení vložení nebo chybové hlášení z databáze. Návrh počítá s tím, že elementární chyby odhalí již klient (například nevyplnění povinných položek), aby na straně klienta nedocházelo ke zbytečným prodlevám. Pokud je neodhalí, bude na nesrovnalosti upozorněn databázovým serverem.

Triggery a integritní omezení databázového serveru by se měly postarat také o zachování konzistence dat. Tím je myšleno například to, že nedovolí odstranit projekt, který obsahuje nedokončené úkoly nebo že při odstranění uživatele

ze skupiny dojde ke smazání záznamu z tabulky, která definuje práva uživatele v dané skupině.

Jednotlivé části programu mají k databázi odlišný vztah a různými způsoby s ní pracují. Aplikační server, respektive vrstva přístupu k datům, disponuje přímým spojením se serverem databázovým, pokládá databázi dotazy, přijímá od ní data, případně deleguje klientovi chybové hlášky, které databáze vrátí. První vrstvou u klienta, která je schopná poznat, že jde o data z databáze (na rozdíl od vrstev komunikačních), je vrstva přístupu k datům. Ta databázové objekty vybaluje z komunikačních zpráv v MIME formátu a předává je poskytovateli dat. Ten je shromažďuje a zprostředkovává jednotlivým modulům. Pluginy tedy s databází komunikují pouze zprostředkovaně. Struktura databáze je pevně určená, navržená na základě analýzy funkcí, které byly pro groupware stanoveny jako zásadní. Pluginům tedy není umožněno databázi měnit či přidávat nové entity. Pokud by z nějakého důvodu bylo nutné další entity přidávat, muselo by dojít k úpravě vrstev poskytování dat u klienta i serveru. Do jiných vrstev by úprava nezasáhla, s výjimkou rozhraní, sloužícího pro komunikaci s pluginy. Přidáním nových položek do tohoto rozhraní by však kompatibilita se staršími pluginy zůstala zachována. V této situaci poprvé vyvstává otázka, zda se vůbec najdou moduly, které by zlepšily funkčnost a použitelnost aplikace a přitom nepřidaly novou entitu do databáze. Inspiraci lze nalézt v provedené analýze groupware aplikací. Jde například o moduly, určené výhradně ke správě souborů, přehledy, umožňující detailní práci s úkoly, které dostal uživatel k vypracování či moduly, které kombinují funkčnost již implementovaných pluginů a umožní odlišný způsob práce s programem.



Obrázek 4 - Zjednodušený diagram databáze bez vyznačených atributů

#### 4.4.1 Hlavní entity

Hlavní entity mají elementární význam pro zajištění toho, aby aplikace byla využitelná tak, jak bylo definováno v kapitole 2.5.

- **Workgroup** - rozděljuje uživatele do skupin, které nemusí být disjunktní. Uživatel tedy může pracovat v různých skupinách zároveň.
- **Project** - shromažďuje informace o projektech. Každý projekt je zařazen do jedné pracovní skupiny, jejíž uživatelé se mohou podílet na jeho realizaci.

- `User` - obsahuje základní údaje o uživateli aplikace.
- `Group` - definuje oprávnění, která mohou mít uživatelé v pracovních skupinách.
- `WGU` (`Workgroup`, `Group`, `User`) - určuje ternární vztah mezi uživatelem, pracovní skupinou a oprávněními, která má uživatel v dané skupině.
- `Task` - popisuje úkoly, ze kterých projekty sestávají. Každý úkol je částí právě jednoho projektu, vztah tedy není nutné realizovat další tabulkou.
- `Journal` – obsahuje záznamy s příspěvkem uživatele
- `Appointment` – obsahuje uživatelem definované události zobrazované v kalendáři
- `User_Project` - určuje uživatele, kteří se podílí na daném projektu. Může se zdát, že informace z této tabulky jsou nahraditelné dotazem na řešitele úkolů, ze kterých projekt sestává. Tato tabulka ovšem slouží k jiným účelům. Umožňuje, aby mohli být uživatelé k projektu přiřazováni předem, tedy ještě před tím, než dojde k dekompozici projektu na jednotlivé úkoly. Vedoucímu projektu, který standardně nemůže k projektu přiřazovat nové uživatele, bude umožněno úkoly nejdříve naplánovat a až pak jim přiřadit řešitele, které má k dispozici

#### 4.4.2 Doplňkové entity

Jde o tabulky, které umožňují rozšířit funkce programu.

- `Company` – zastřešuje uživatele, pracovní skupiny a projekty jedné společnosti, která využívá služby aplikace. Jde o velmi jednoduchou tabulku, která kromě identifikátoru a jména společnosti neobsahuje nic jiného. Informace o společnostech nejsou nikam delegovány, slouží pouze ke zpřehlednění správy programu a k omezení domény, ve které může správce pracovních skupin vytvářet nové skupiny.
- `Project_Note`, `Task_Note` - uchovávají poznámky připisované k projektům (resp. úkolům). Poznámku k projektu může psát libovolný uživatel, který se na projektu podílí. Tabulka `Project_Note` musí tedy navíc obsáhnout informaci o tom, který uživatel poznámku psal. Poznámky k úkolu vyplňuje pouze jeho řešitel.
- `Project_File`, `Task_File` - obsahují informace o souborech, přiložených k projektům (resp. úkolům), například název, umístění na serveru nebo velikost v bytech.
- `Task_Dependency` - u každého úkolu určuje, na kterých dalších úkolech z projektu je daný úkol závislý.

## 5 Implementace

Kapitola popisuje konkrétní implementaci aplikace, včetně popisu tříd a významných metod. Kromě charakteristiky aplikačního serveru a klientské části je prostor věnován také knihovnám, které jsou využívány oběma částmi programu. Knihovny modelují datové objekty, které slouží k reprezentaci dat, získaných z databáze. Tyto objekty jsou ve zbylých částech programu využívány natolik, že je popis této knihovny zařazen na prvním místě. Následuje popis aplikačního serveru a nakonec klientské části.

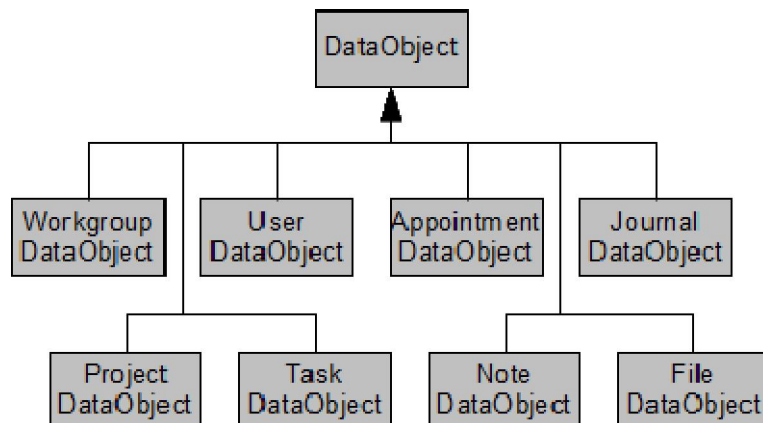
### 5.1 Datové objekty

Knihovna `DataObjects` je tvořena několika třídami. Jednotlivé třídy slouží k reprezentaci dat v nejvyšších vrstvách klientské a serverové části aplikace. Konkrétně jde o vrstvy poskytování dat. Na aplikačním serveru jsou tyto objekty vytvářeny z dat, získaných z databáze, následně jsou serializovány a v této podobě odeslány klientovi, který je v odpovídající vrstvě deserializuje a dále s nimi pracuje.

Implementace datových objektů splňuje kritéria, kladená návrhem aplikace:

- Program pracuje s daty odlišných typů, a proto musí být pro nejvyšší vrstvy snadné jednotlivé typy rozlišit. Z toho důvodu každému typu dat odpovídá třída, definující položky, které jsou pro daný typ specifické. Typem dat je myšlen například projekt, úkol, záznam o uživateli, záznam o souboru a další.
- Nižší vrstvy, které slouží zejména k hromadnému zpracovávání dat (odesílání či příjmu), by naopak neměly na typ dat brát ohled. Proto jsou všechny datové objekty potomkem jediného předka `DataObject`. Ty části programu, pro které není typ dat určující, pracují polymorfně s tímto předkem. Tím je jejich implementace jednodušší a přidání nového datového typu neovlivní zbytek implementace.
- Třetím požadavkem je, aby položky datových typů odpovídaly položkám dat z databáze, které reprezentují a zároveň aby serializované datové objekty měly podobu některého z otevřených formátů, popsanych v předešlé kapitole. Toho je docíleno předefinováním metody `ToString()`, kterou je možné volat u každého objektu. V textové podobě tak mají objekty stejný tvar, jaký používá server Kolab.
- Definice datových objektů je jednoduchá a přímočará. Díky tomu je možné v budoucnu vytvořit nový datový typ, se kterým bude aplikace pracovat. Jedinou podmínkou je, aby byl nový objekt potomkem předka `DataObject`. V opačném případě nebude rozpoznán nižšími vrstvami. Jak je patrné z obrázku 5, ve stávající implementaci jsou všechny třídy, popisované v následující kapitole, jejím přímým potomkem.

## 5.1.1 Přehled tříd, jejich položek, vlastností a metod



Obrázek 5 - Dědičnost tříd v knihovně DataObjects

### 5.1.1.1 Třída DataObject

`DataObject` je předkem všech datových objektů, které jsou v knihovně definovány. Obsahuje jedinou položku `uid`, což je unikátní identifikátor, který má každý datový objekt.

### 5.1.1.2 Třída WorkgroupDataObject

Objekt tohoto typu slouží k reprezentaci pracovní skupiny. Obsahuje následující položky:

- `Name` – jméno pracovní skupiny
- `Leaders` – seznam uživatelů (resp. jejich identifikátorů), kteří jsou vedoucími skupiny
- `Users` – seznam uživatelů, resp. identifikátorů, kteří jsou zařazeni do dané skupiny
- `Access` – určuje vztah uživatele, kterému jsou data zasílána, k pracovní skupině, kterou objekt reprezentuje.

### 5.1.1.3 Třída ProjectDataObject

Instance této třídy odpovídá záznamu o projektu. Metoda `ToString()` z objektu vytvoří textový záznam, který je ve formátu `iCalendar`. Je popsán v kapitole 4.2.2.

Položky objektu:

- `Description` – popis projektu
- `End_date` – datum, do kterého musí být projekt ukončen

- `Finished` – procentuální vyjádření stavu dokončení projektu. Nutno podotknout, že tuto položku není možné měnit, počítá se automaticky ze stavů dokončení jednotlivých úkolů, ze kterých projekt sestává.
- `Leader` – identifikátor uživatele, který je vedoucím projektu
- `Start_date` – datum začátku projektu
- `Summary` – název, shrnutí projektu
- `Tasks` – seznam identifikátorů úkolů, které do projektu náleží
- `Users` – seznam uživatelů, kteří se na projektu podílejí
- `Workgroup` – pracovní skupina, do níž projekt patří

#### 5.1.1.4 Třída `TaskDataObject`

`TaskDataObject` uchovává informace o úkolech. Jeho textová podoba je opět ve formátu iCalendar.

Položky:

- `Dependencies` – obsahuje identifikátory úkolů ze stejného projektu, na kterých úkol závisí
- `Description` – popis úkolu
- `End_date` – datum, do kterého musí být úkol hotový
- `Finished` – procentuální vyjádření stavu dokončení úkolu
- `Project` – identifikátor projektu, ke kterému je úkol zařazen
- `Start_date` – datum, kdy úkol začíná
- `Summary` – název, shrnutí úkolu
- `User` – uživatel, který úkol řeší

#### 5.1.1.5 Třída `UserDataObject`

Jde o objekt, který obsahuje informace o uživateli aplikace. Zavoláním metody `ToString()` u instance tohoto objektu je vygenerován textový řetězec ve formátu vCard. Položky:

- `Address` – adresa uživatele
- `Admin` – booleovská hodnota určující, zda je uživatel administrátorem v rámci společnosti
- `E-mail` – e-mailový kontakt
- `Full_name` – celé jméno, tato položka slouží k přehlednému zobrazování jména v seznamech uživatelů
- `Given_name` – křestní jméno

- `Last_name` - příjmení
- `Nick_name` – unikátní přihlašovací jméno do aplikace
- `Phone` – telefonní číslo

#### 5.1.1.6 Třída `AppointmentDataObject`

Další objekt, který má textovou podobu formátu iCalendar, Je určen k reprezentaci kalendářových událostí, definovaných uživatelem. Položky:

- `Date` – datum události
- `Description` – popis
- `Summary` – název události

#### 5.1.1.7 Třída `JournalDataObject`

Tento objekt odpovídá příspěvku do žurnálu, v textové podobě jde o formát `Journal`, který používá Kolab.

Položky:

- `Date` – datum, kdy byl příspěvek napsán
- `Text` – text příspěvku
- `Title` – titulek příspěvku
- `User` – identifikátor uživatele, který je autorem příspěvku
- `Workgroup` – pracovní skupina, ve které je příspěvek publikován

#### 5.1.1.8 Třída `FileDataObject`

Instance tohoto objektu nese informace o souboru, jehož stažení si může uživatel ze serveru vyžádat.

Položky:

- `Eid` – identifikátor projektu nebo úkolu, ke kterému je soubor přiřazen
- `Extension` – přípona souboru
- `Name` – jméno souboru
- `Path` – cesta k souboru
- `Size` – velikost souboru v bytech

#### 5.1.1.9 Třída `NoteDataObject`

Slouží k uchování poznámky k projektu nebo úkolu. Textová podoba je stejná jako u Kolabu.

Položky:

- `Date` – datum, kdy byla poznámka napsána
- `Eid` – identifikátor projektu nebo úkolu, ke kterému poznámka patří
- `Text` – vlastní text poznámky
- `User` – uživatel, který poznámku napsal

## 5.2 Aplikační server

Server je rozdělen do dvou vrstev – komunikační a přístupové. Každé z vrstev odpovídá jedna třída, která implementuje funkce, popsané v návrhu serveru. Vztah mezi vrstvami je znázorněn na obrázku 2.

### 5.2.1 Třída `DataAccessLayer`

Třída implementuje vrstvu přístupu k datům, která zajišťuje komunikaci s databázovým serverem. Přístup do databáze je realizován pomocí nástroje LINQ. To znamená, že v rámci aplikace jsou vytvořeny datové struktury, odpovídající databázovým tabulkám, záznamům, které jsou v tabulkách uloženy a metodám, které jsou implementovány v rámci databáze. Tato struktura představuje takzvaný datový kontext, který je velmi těsně spjatý s databází. Ve skutečnosti tedy třída `DataAccessLayer` nepracuje přímo s databází, ale s instancí datového kontextu, který realizuje vlastní databázové dotazy.

Datový kontext je využíván ve všech metodách, které pracují s databází. Je to tedy jedna z elementárních proměnných třídy. Druhou takovou proměnnou představuje reference na komunikační vrstvu, díky které je možné odesílat klientům data.

Ještě před popisem jednotlivých metod je vhodné uvést třídu `ClientInfo`, která je sdílena oběma vrstvami serveru a obsahuje informace o připojeném klientu. Informace jsou relevantní zejména pro komunikační vrstvu, podrobněji bude proto třída rozebrána až v kapitole o této vrstvě. Nyní je zmíněna kvůli tomu, že slouží jako identifikátor připojeného klienta v průběhu vyřizování jeho požadavků.

Metody třídy `DataAccessLayer` (v závorce jsou uvedeny parametry)

- `ProcessMessage (string command, ClientInfo client)` – metodu volá komunikační vrstva, pokud obdrží zprávu, určenou vrstvě vyšší. Úkolem této metody je zprávu identifikovat. V první řadě může jít o požadavek na zaslání dat z databáze – projektů, úkolů, uživatelů a dalších. Druhým typem požadavku je žádost o stažení nebo nahrání souboru. Třetím typem zprávy, která může serveru přijít, jsou data, určená k vložení do databáze. Poslední akcí, kterou metoda rozlišuje, je pokus uživatele o přihlášení – jde tedy o ověření uživatelského jména a hesla. Parametr `command` je textový řetězec – příchozí zpráva, parametr `client` identifikuje klienta, který zprávu zaslal.

- `Appointments, Files, Journals, Notes, Projects, Tasks, Users (ClientInfo client)` – jde o sedm metod, které jsou volány metodou `ProcessMessage`. Každá z nich získává z databáze data, z těchto dat vytváří příslušné datové objekty třídy `DataObjects` a předává je metodě `Send()` k odeslání.
- `FileDownloadRequest (string command, ClientInfo client)` – metoda je zavolána v případě, že klient vyžaduje stažení souboru ze serveru. Parametr `command` obsahuje identifikátor souboru. Metoda díky němu z databáze zjistí cestu k souboru (klient ji nezná), ověří, že přihlášený uživatel má právo přístupu k souboru a iniciuje zaslání souboru – zavolá příslušnou metodu komunikační vrstvy.
- `FileUploadRequest (string command, ClientInfo client)` – tato metoda je zavolána, pokud se klient rozhodne nahrát na server soubor. Nejprve ověří, zda je žádost oprávněná, následně vygeneruje cestu, kam soubor uložit a předá ji komunikační vrstvě se svolením, že je možné soubor nahrát. Pokud je operace úspěšná, vytvoří v databázi o souboru záznam. Parametr `command` obsahuje příslušné vlastnosti souboru jako jméno, velikost či identifikátor projektu, se kterým má být asociován.
- `NewDataRequest (string command, ClientInfo client)` - metoda slouží k rozbalení zprávy ve formátu MIME, která obsahuje nová data určená k uložení do databáze, zasláná klientem. Využívá k tomu knihovnu, určenou pro práci se zprávami MIME. Dle typu obsahu zprávy rozliší, o jaký typ dat se jedná, vytvoří z obsahu zprávy příslušný datový objekt třídy `DataObject` a předá jej metodě, která se postará o vložení dat do databáze.
- `UpdateNewDelete (DataObject o, boolean? update, ClientInfo client)` – prvním parametrem této metody je datový objekt, se kterým bude metoda pracovat. Druhý parametr `update` nabývá tří hodnot a určuje, zda jde o vložení datového objektu do databáze, smazání či aktualizaci (`true` – aktualizace objektu, `false` – nový objekt, `null` – smazání objektu z databáze). Všechny tyto akce metoda také implementuje, a to s objekty všech typů.
- `Send (string data, string contentType, ClientInfo client)` – metoda dostane objekt v textové podobě (parametr `data`), může jít například o formát `vCard` nebo jiný z výše popsaných. Dalším parametrem je `contentType`, tedy typ obsahu, který odpovídá jednomu z typů, které využívá Kolab server pro rozlišení, o jaká data jde. Metoda z daných parametrů vytvoří zprávu ve formátu MIME (opět díky zmíněné knihovně) a předá ji komunikační vrstvě, která se postará o její odeslání.
- `SendMessage (string message, ClientInfo client)` – metoda je využívána v případě, že je nutné zaslat klientovi jinou zprávu, než ve formátu MIME. Typicky může jít o nějaký příkaz protokolu IMAP.

Metoda pouze předá řetězec `message` komunikační vrstvě, která jej odešle.

Následuje několik metod, které slouží k administraci aplikace. Jsou volány pouze správcem programu v situacích, kdy je nutné vytvořit v databázi prostor pro novou společnost, vytvořit ve společnosti první uživatelský účet (s rolí administrátora pracovních skupin) nebo naopak všechny záznamy o společnosti odstranit. Tyto metody mohou být volány pouze lokálně z konzole serveru.

- `NewCompany (string name)` – vytvoří v databázi záznam o nové společnosti, vrací logickou hodnotu, značící, zda operace proběhla správně.
- `DeleteCompany (string name)` – odstraní z databáze společnost zadaného jména.
- `NewUser (string name, string surname, string login, string company, string password)` – metoda, umožňující vytvoření nového uživatelského účtu v rámci společnosti. Je určena k vytvoření prvního uživatelského účtu, aby mohla společnost software začít používat.

## 5.2.2 Třída `CommunicationLayer`

Třída implementuje komunikační vrstvu, která obstarává spojení klientů s aplikačním serverem. Jejimi hlavními úkoly tedy jsou inicializace a ukončení spojení s klientem, výměna zpráv s vyšší komunikační vrstvou a realizace přenosu souborů v obou směrech.

Klientů může být k serveru připojeno více. K jejich identifikaci slouží třída `ClientInfo`. Jde o jednoduchou třídu s několika položkami:

- `Socket` – soket, na kterém probíhá komunikace mezi serverem a klientem, který instance třídy `ClientInfo` reprezentuje
- `Buffer` – pole bytů, do kterého jsou načítána příchozí data
- `MemoryStream` – stream, sloužící k příjmu a odesílání souborů
- `receiveBufferSize` – konstanta, určující velikost přijímacího bufferu, typicky je její hodnota 1024
- `name` – nepovinná položka se jménem klienta

Komunikace je implementována pomocí socketů, probíhá asynchronně, tedy bez nuceného čekání na spojení a příjem zpráv. V konstruktoru třídy se otevře serverový socket, o další průběh komunikace se starají následující metody:

- `AcceptCallback` – metoda je zavolána vždy, pokud se připojí nový klient. O její zavolání se stará delegát, definovaný u serverového socketu. Po připojení nového klienta se vytvoří instance třídy `ClientInfo` s příslušnými parametry, dále nový klientský socket, na kterém bude komunikace s tímto klientem probíhat a tento socket začne asynchronně naslouchat.

- `ServerReceiveCallback` – tato metoda vyřizuje veškeré zprávy, které klient zasílá. Opět je volána delegátem, přiřazeným příslušnému klientskému socketu. Metoda rozlišuje zprávy na servisní a komunikační. Komunikační zprávy rozbalí a jejich obsah předá nadřazené vrstvě, na servisní zprávy příslušně reaguje. Může jít o pozdrav klienta po připojení, zprávu, která testuje spojení, oznámení o odpojení klienta či zprávu žádající o inicializaci zaslání datového proudu se souborem.
- `DisconnectClient (ClientInfo client)` – metoda je volána v případě, že dojde k odpojení klienta. Odpojení je iniciováno vždy klientem, buď po explicitním oznámení, nebo odchycením výjimky.
- `Send (string message, char type, ClientInfo client)` – metoda slouží k posílání zpráv klientovi `client`. Znak `type` označuje typ zprávy, do řetězce `message` je možné vložit doplňující parametry.
- `SendMessage (string message, ClientInfo client)` – jde o metodu, která je veřejně přístupná a slouží vyšším vrstvám k zasílání zpráv. Umožňuje zaslat pouze komunikační zprávu.
- `OnSendMessage` – metoda je volána asynchronně po odeslání zprávy, slouží k ukončení odesílání, které bylo při posílání zprávy zahájeno.
- `ReceiveStream (Stream stream, long Length, ClientInfo client)` – slouží k příjmu datového proudu `stream` délky `long`, který zasílá klient.
- `SendStream (Stream stream, ClientInfo client)` – odesílá klientovi `client` datový proud `stream`.
- `OnClosing` – metoda je volána při ukončení aplikace. Pokouší se zavřít všechny otevřené sokety.

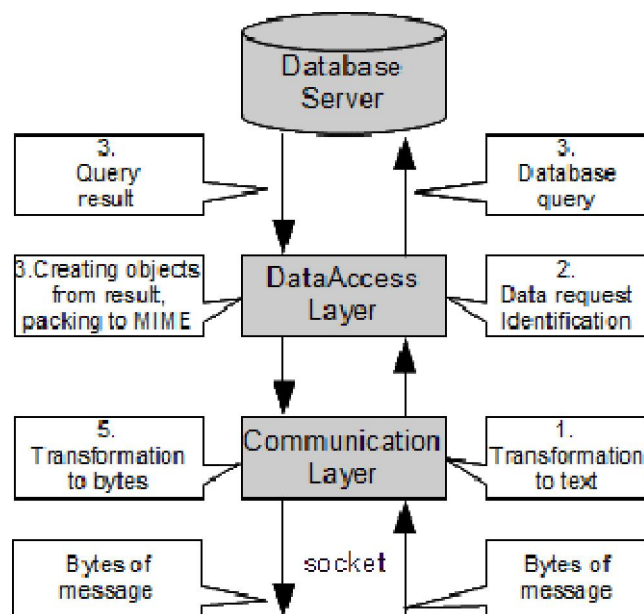
### 5.2.3 Zpracování dat

Následující dvě kapitoly popisují dvě nejčastější operace, které server provádí. Je na nich ilustrováno, jakým způsobem se data v jednotlivých vrstvách transformují.

#### 5.2.3.1 Žádost o data z databáze

1. Socket komunikační vrstvy ve třídě `DataAccessLayer` přijme od klienta data v podobě několika bytů. Dekóduje je do textové podoby a zjistí, že jde o zprávu, určenou vyšší vrstvě. Zbaví ji hlavičky a zbytek textového řetězce předá metodě `ProcessMessage` nadřazené vrstvy `DataAccessLayer`.
2. Metoda `ProcessMessage` identifikuje, že jde o žádost o zaslání určitého typu dat (například projektů) a deleguje ji odpovídající metodě (v případě projektů jde o metodu `Projects()`). Metody jsou rozlišeny dle typu dat, o která klient žádá.

3. Metody, přístupující k datům, disponují referencí na datový kontext. Pomocí něj získají vhodným dotazem z databáze data, která se týkají přihlášeného uživatele (například záznamy o projektech, na kterých pracuje). Díky typově jednoznačnému jazyku LINQ, který je použitý k dotazování, lze z výsledků dotazů přímo vytvářet objekty z knihovny DataObjects. Metoda Projects() tedy výsledek dotazu na projekty transformuje do seznamu objektů typu ProjectDataObject, kde každý prvek seznamu představuje jeden záznam z databáze. Nakonec metoda předá postupně všechny prvky seznamu metodě Send().
4. Metoda Send() přijímá parametrem objekt typu DataObject, serializuje jej do textové podoby a připojí jej jako přílohu MIME zprávy. Na základě typu objektu označí typ obsahu zprávy stejně jako server Kolab. Přílohu s projektem identifikuje řetězec text/calendar. MIME zpráva, která má textový charakter, je nakonec předána nižší komunikační vrstvě (třídě CommunicationLayer), konkrétně metodě SendMessage() této vrstvy.
5. Metoda SendMessage() připojí k řetězci, který jí byl předán parametrem, identifikační hlavičku komunikační vrstvy, transformuje zprávu do pole bytů a to odešle klientovi.



**Obrázek 6 - Průběh zpracování žádosti o data**

### 5.2.3.2 Vložení nových dat do databáze

1. První krok je stejný jako v předchozím případě. Komunikační vrstva obdrží zprávu a předá ji metodě `ProcessMessage()` vyšší vrstvy.
2. `ProcessMessage()` rozpozná řetězec jako zprávu ve formátu MIME a předá jej metodě `NewDataRequest()`.
3. Metoda `NewDataRequest()` identifikuje jednotlivé položky MIME zprávy a dle typu obsahu rozezná data, která jsou v příloze zprávy obsažena a typ akce, která se má s daty provést (odstranění, úprava, nová data). Může se jednat například o nový projekt, reprezentovaný textovým řetězcem ve formátu iCalendar. Na základě typu obsahu vytvoří ze záznamu objekt z knihovny `DataObjects` a předá jej metodě `UpdateNewDelete()`, která provádí úpravy v databázi.
4. `UpdateNewDelete()` nejprve ověří, zda má uživatel oprávnění k provedení dané akce a pokud ano, provede zásah do databáze. Na závěr prostřednictvím metody `SendMessage()` komunikační vrstva odešle klientovi výsledek akce. Může být pozitivní nebo negativní.

## 5.3 Klientská část

Implementace reflektuje požadavek modularity, snadné rozšiřitelnosti o další pluginy a přehlednosti grafického prostředí. Jádro aplikace je tvořeno několika vrstvami, moduly jsou jednoduché a slouží k práci s konkrétním typem dat. Způsob, kterým jádro aplikace poskytuje modulům data, je poměrně specifický, a proto je mu věnována samostatná část hned v úvodu kapitoly. Pro lepší pochopení struktury programu slouží obrázek 3.

### 5.3.1.1 Poskytování dat

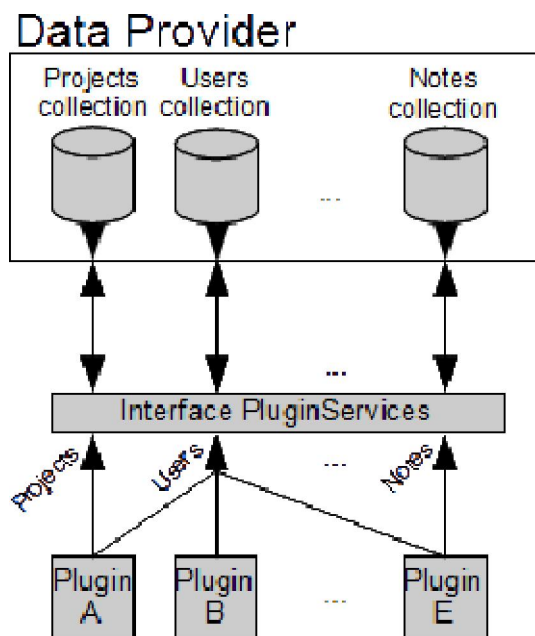
Datové struktury, určené ke shromažďování dat zaslaných ze serveru, jsou situovány ve třídě `DataProvider`. Slouží k uchování kolekcí datových objektů, které reprezentují data z databáze. Pokud modul žádá o data určitého typu (například projekty), vytvoří se pro tento typ dat specifická kolekce. Při používání programu se tedy ve třídě `DataProvider` nachází několik takových kolekcí, přičemž jedna může být určena k ukládání datových objektů s projekty, další s uživateli a podobně.

Při návrhu třídy bylo nutné zohlednit to, že různé moduly mohou požadovat stejné kolekce dat. Například kolekce uživatelů je využívána modulem Kontakty a zároveň Správou pracovních skupin. Nebylo by tedy rozumné požadovat stejná data z databáze dvakrát či vícekrát pro každý modul a vytvářet více stejných kolekcí. To by způsobilo i další komplikaci – moduly mohou data pozměnit či smazat. V případě, že by každý modul disponoval vlastní kolekcí s daty, bylo by nutné

udržovat strukturu existujících kolekcí stejného typu dat a v případě změny upravit každou z nich.

K uchování datových objektů slouží tzv. pozorovatelná kolekce, vytvořená pro účely programu. Pro každý typ dat existuje pouze jedna taková kolekce, a pokud modul požaduje data nějakého typu, je mu vrácena reference na příslušnou kolekci. To znamená, že pokud moduly požádají o všechny typy dat, které mohou požadovat, ve třídě `DataProvider` bude právě osm kolekcí (kolekce pracovních skupin, uživatelů, projektů, úkolů, událostí v kalendáři, položek žurnálu, poznámek a souborů) a tento počet se již nebude měnit.

Grafické prvky většiny pluginů jsou na kolekce přímo napojeny, čímž se značně zjednodušuje práce se zobrazováním dat. Pokud dojde v kolekci ke změně, například ze serveru přijde nový datový objekt odpovídajícího typu, moduly nemusejí být o změně explicitně informovány. V tom spočívá pozorovatelnost kolekce.



Obrázek 7 - Přístup pluginů ke kolekcím dat

Zažádá-li modul o kolekci s daty nějakého typu, vždy získá referenci na soubor všech objektů onoho typu, ke kterým má přihlášený uživatel přístup. Ovšem moduly někdy nemanipulují s celou množinou objektů, ale pouze s vybranými prvky dle nějakého kritéria, filtru. S tímto problémem se třída `DataProvider` vyrovnat nedokáže. Řešením by bylo vytvoření více kolekcí dat jednoho typu, čímž by nastal problém s narůstajícím počtem kolekcí, popsany výše. Situací se zabývají až samotné moduly. Potřebuje-li modul pracovat s kolekcí, obsahující pouze podmnožinu všech objektů daného typu, vytvoří si lokální instanci

kolekce a k hlavní kolekci s objekty přidá referenci na metodu, která je volána vždy, když v hlavní kolekci dojde ke změně. Metoda obsahuje filtr, který rozhodne, zda je změněný objekt pro lokální kolekci relevantní či nikoli a na základě tohoto rozhodnutí upraví nebo neupraví obsah lokální kolekce. Tím zůstane počet hlavních kolekcí s daty omezen počtem typů dat, které se v programu používají.

Princip vytváření lokálních kolekcí se může zdát zbytečně složitým a těžkopádným, volba tohoto systému však vychází z několika poznatků. Poskytování dat je navrženo s ohledem na snadné rozšíření aplikace o další moduly. V jádru aplikace jsou pouze kolekce, odpovídající hlavním tabulkám databáze (respektive hlavním typům dat). Lokální kolekce, které si drží jednotlivé moduly, do jisté míry představují výsledky různých databázových dotazů, filtrujících řádky tabulky. Jsou to kolekce dat, z nichž všechny prvky splňují nějakou podmínku. Není v silách třídy `DataProvider`, aby implementovala a konkrétními metodami zpřístupnila všechny možné kolekce dat, splňujících nějakou podmínku. Tím, že je tato problematika přesunuta do jednotlivých modulů a je vyřešena možností přidat k hlavní kolekci filtr, je implementace třídy `DataProvider` jednoduchá, přehledná a snadno rozšiřitelná o další typy dat. Důležitý je poznatek, že každý objekt se v paměti nachází pouze jednou, veškeré operace manipulují pouze s referencemi. Implementace je tedy rychlá a prostorově nenáročná.

Poslední důležitou vlastností je způsob, kterým jsou kolekce s daty plněny. Opět je využito toho, že kolekce jsou pozorovatelné. V okamžiku, kdy modul požaduje data, může dojít ke dvěma situacím. Buď je kolekce naplněná, neboť již data daného typu byla někdy vyžadována. V takovém případě je modulu vrácena reference na tuto kolekci. Nebo je modul prvním, který o data daného typu žádá. Pak je vytvořena nová kolekce, na server je zaslána zpráva s žádostí o příslušná data a modulu je vrácena reference na novou kolekci. Je zřejmé, že v tom okamžiku bude kolekce ještě prázdná, neboť nějakou dobu trvá, než data ze serveru dorazí. Díky vlastnostem pozorovatelné kolekce není nutné na nová data čekat a aplikace může fungovat bez omezení. Data mohou přicházet se zpožděním a tak, jak ke klientovi přicházejí, jsou zařazována do odpovídajících kolekcí. To je veškerá režie, kterou aplikace s příchozími daty má. Vložení nového objektu do pozorovatelné kolekce je objekt okamžitě k dispozici všem modulům, které mají na kolekci referenci, a je zobrazen ve všech grafických prvcích, které jsou na kolekci napojeny.

Pro lepší ilustraci problematiky je v kapitole 5.3.5 popsáno několik základních operací, které program s daty vykonává. Pro jejich pochopení je ovšem nutné znát vnitřní strukturu aplikace, uvedenou v následujících kapitolách.

### 5.3.2 Knihovna `PluginInterface`

`PluginInterface` je knihovna, sdílená jádrem aplikace a všemi moduly. Obsahuje definici dvou rozhraní. To první – `IPlugin` – musí implementovat každý modul. Druhé rozhraní `IPluginHost` implementuje jádro aplikace. Knihovnu sdílí všechny části klientského programu, obsahuje proto také deklaraci pozorovatelné kolekce.

### 5.3.2.1 Interface IPlugin

Jde o rozhraní, které implementuje každý modul. Definuje následující položky a metody:

- `IPluginHost Host` – položka obsahuje referenci na jádro aplikace, zpřístupňuje modulům funkce programu
- `string Name, Description, Author, Version` – obsahují po řadě informace o jménu, popisu, autorovi a verzi pluginu.
- `BitmapSource Icon` – ikona modulu, která se zobrazí v menu aplikace
- `MenuItem Menu` – položka nástrojové lišty s menu, které může plugin používat
- `UserControl MainInterface` – hlavní prvek grafického rozhraní, který se zobrazí, když se plugin inicializuje
- `Initialize ()` – metoda, volaná po spuštění modulu
- `Dispose ()` – metoda, volaná při ukončení modulu

### 5.3.2.2 Interface IPluginHost

Rozhraní umožňuje pluginům požadovat data, měnit je či mazat, případně iniciovat stažení souboru.

- `Bool Online` - ukazuje, zda je aplikace připojena k serveru
- `Workgroups(), Projects(), Tasks(), Users(), Notes(), Files(), Appointments(), Journals()` – metody, které vrací referenci na pozorovatelnou kolekci s příslušným typem dat (po řadě pracovní skupiny, projekty, úkoly, uživatelé, poznámky, soubory, události v kalendáři, položky žurnálu)
- `NewDataObject, UpdateDataObject, DeleteDataObject (DataObject o)` – metody, které do databáze vloží nová data nebo aktualizují či odstraní existující záznamy. Přesněji řečeno tyto metody iniciují odeslání dat na server, který se o vložení do databáze.
- `GetFile (FileDataObject file)` – metoda, která iniciuje stažení souboru `file` ze serveru
- `UploadFile (FileDataObject file)` – metoda, která iniciuje nahrání souboru `file` na server

### 5.3.2.3 Třída DataCollection

Tato třída implementuje pozorovatelnou kolekci. Elementární položkou kolekce je generický seznam, který je schopný shromažďovat objekty typu `DataObject`, tedy veškerá data, se kterými program pracuje. Jde o spojový seznam, který ovšem neumožňuje rychle vyhledávat. Z toho důvodu jsou reference

na datové objekty udržovány také v hašovací tabulce, která je na základě znalosti identifikátoru schopná přistoupit ke konkrétnímu objektu v konstantním čase.

Přístup k seznamu položek zajišťují metody `Add (DataObject data)`, `Remove (string id)` a `ReturnDataObject (string id)`, které slouží po řadě k vložení nového objektu do kolekce, odstranění objektu daného `id` z kolekce a vrácení konkrétního objektu, pokud se v kolekci nachází. Všechny metody umožňují bezpečně operovat s kolekcí různým vláknům.

Poslední položkou je delegát `OnChanged`. Obsahuje reference na metody, které se mají zavolat při změně kolekce. Tento delegát slouží modulům k tomu, aby si mohly vytvořit lokální instanci datové kolekce a v ní mít pouze podmnožinu objektů určitého typu.

### 5.3.3 Jádru programu

Implementace odpovídá návrhu (viz kapitola 4.3.1) – jádro sestává ze čtyř vrstev, každá vrstva je má podobu jedné třídy. Následuje popis jednotlivých tříd v pořadí od nejvyšší vrstvy.

#### 5.3.3.1 Třída `PluginServices`

Třída implementuje rozhraní `IPluginHost` a obsahuje reference na dvě nižší vrstvy: Na poskytovatele dat a na vrstvu přístupu k datům. První referenci využívá k získávání kolekcí s daty, které předává pluginům. Druhá reference slouží ke zprostředkování zasílání souborů a kodesílání nových a aktualizovaných dat na server. To jsou funkce, kterými se poskytovatel dat nezabývá. Třída implementuje tyto metody rozhraní `IHost`:

- `Workgroups()`, `Projects()`, `Tasks()`, `Users()`, `Notes()`, `Files()`, `Appointments()`, `Journals()` – každá z těchto metod zavolá stejnojmennou metodu třídy `DataProvider`, která vrátí referenci na datovou kolekci s objekty příslušného typu
- `NewDataObject`, `UpdateDataObject`, `DeleteDataObject (DataObject o)` – tyto metody předávají datový objekt `o`, který dostaly jako parametr, vrstvě přístupu k datům, která jej odešle databázi. Všechny tři metody vrací logickou hodnotu, která je splněna v případě, že server data akceptuje.
- `GetFile (FileDataObject file)`, `UploadFile (FileDataObject file)` – slouží k inicializaci přenosu souboru ze serveru nebo na server a starají se o zadání potřebných dat. Tím je myšlena například cesta k souboru, který se má nahrát.

Třída `PluginServices` neslouží výhradně pluginům, ale implementuje také veškeré služby, spojené se zajištěním modularity.

- `Types.AvailablePlugins AvailablePlugins` – proměnná, obsahující kolekci modulů, nalezených po startu aplikace

- `FindPlugins (string path)` – metoda, která prohledá složku s názvem `path` a najde všechny knihovny, které implementují rozhraní `IPlugin` a jsou tedy vhodnými moduly pro aplikaci
- `ClosePlugins ()` – metoda, zavírající instance pluginů při ukončení aplikace
- `AddPlugin (string FileName)` – pomocná metoda, která přidá nalezený plugin (soubor s názvem `FileName`) do kolekce `AvailablePlugins` a vytvoří novou instanci pluginu.

### 5.3.3.2 Třída `DataProvider`

`DataProvider` implementuje vrstvu, která zajišťuje, aby moduly měly k dispozici data. Princip poskytování dat je popsán v úvodu této kapitoly. Zbývá doplnit, jakým způsobem je realizováno poskytování dat bez připojení k serveru, což je jeden z požadavků na tuto třídu.

Pokud modul žádá data, pro která ještě není vytvořena kolekce, vytvoří se nová instance kolekce a reference na ni se předá modulu. V tu chvíli se `DataProvider` začne zabývat naplněním kolekce daty. Pokud je spojení se serverem funkční, zprostředkuje odeslání zprávy s žádostí o data. V opačném případě program zkontroluje, zda jsou na pevném disku uložena data, která by mohla být použita a pokud ano, načte je.

Před ukončením práce s programem jsou data stažená ze serveru automaticky uložena na disk, aby mohla být k dispozici při dalším spuštění, pokud by nebylo možné se připojit k serveru. Pokud se během práce offline podaří navázat spojení se serverem, `DataProvider` data v kolekcích aktualizuje.

Třída využívá služeb nižší vrstvy, konkrétně vrstvy poskytování dat. Předává jí požadavky na data, která mají být serverem zaslána. Kolekce datových objektů jsou soustředěny v hašovací tabulce. Na základě typu dat je k dispozici odpovídající kolekce. Vracení referencí na kolekce a vkládání nových objektů tak probíhá s konstantní složitostí. `DataProvider` je navržen tak, aby bylo možné jednoduše přidávat nové datové typy, se kterými program pracuje. Jedinou podmínkou je, aby byly potomkem třídy `DataObject`, deklarované v knihovně `DataObjects`.

Metody:

- `Workgroups(), Projects(), Tasks(), Users(), Notes(), Files(), Appointments(), Journals()` – každá z těchto metod zavolá metodu `GetData()`, které parametrem předá konkrétní datový typ. Metoda `GetData()` vrátí referenci na odpovídající datovou kolekci.
- `GetData (Type type)` – metoda slouží k poskytování referencí na datové kolekce. Pokud kolekce typu `type` neexistuje, vytvoří novou instanci. Následně zkontroluje, zda je k dispozici připojení k serveru. Pokud ano, pomocí nižší vrstvy odešle serveru žádost o zaslání dat zadaného typu. V opačném případě zavolá metodu, která se pokusí načíst odpovídající data z disku.

- `InsertDataObject (DataObject o)` – přijde-li ze serveru datový objekt, je pomocí této metody zařazen do kolekce odpovídajícího typu
- `DeleteDataObject (DataObject o)` – pokud uživatel smaže z databáze záznam, metoda jej odstraní z příslušné kolekce.
- `LoadFromDisc (Type type)` – metoda, která načítá data zadaného typu z pevného disku
- `SaveToDisc ()` – metoda je volána při odhlášení nebo ukončení programu. Ukládá na pevný disk všechna data stažená ze serveru v serializované podobě.

### 5.3.3.3 Třída `DataAccessLayer`

Třída implementuje vrstvu, která realizuje odesílání a příjem dat. Podobně jako stejnojmenná vrstva na serveru pracuje se zprávami ve formátu MIME a využívá k tomu odpovídající knihovnu. Druhým typem zpráv jsou příkazy protokolu IMAP, které slouží ke zprostředkování dalších služeb na serveru. Tím je myšlena například autentizace nebo žádosti o data z databáze.

Třída disponuje referencí na nižší komunikační vrstvu, které předává data k odeslání, na poskytovatele dat, kterému deleguje přijaté datové objekty a nakonec na vrstvu `PluginServices`, které předává informace o zasílání souborů. Dalšími proměnnými jsou řetězec `Login`, který obsahuje přihlašovací jméno přihlášeného uživatele, a booleovskou hodnotu `ConnectionExists`, která slouží vyšším vrstvám k informaci o tom, zda je k dispozici připojení k serveru.

Metody:

- `Authentication (String userName, SecureString password)` – metoda slouží k ověření uživatelského jména a hesla. Je-li program připojen k serveru, řeší autentizaci server. V opačném případě, byl-li uživatel již někdy k programu přihlášen, proběhne ověření zadaných údajů z informací uložených na disku. Poskytovatel dat plní kolekce pouze na základě identifikátoru přihlášeného uživatele a privátní booleovské proměnné, která určuje, zda přihlášení proběhlo v pořádku. To jsou položky třídy `DataAccessLayer` a jejich hodnotu je možné měnit pouze z této metody.
- `DataRequest (string dataType)` – odesílá na server žádost o zaslání dat typu `dataType`
- `FileTransferProgress (int x)` – tato metoda je volána nižší vrstvou, parametrem je procentuální vyjádření objemu přenesených dat souboru. Metoda informuje třídu `PluginServices`, která informaci o průběhu přenosu zobrazuje uživateli.
- `GetFile (FileDataObject file, string localPath)` – iniciuje stažení souboru `file` ze serveru. Parametr `localPath` určuje, kam se má stažený soubor uložit.

- `UploadFile (FileDataObject file)` – iniciuje zaslání souboru `file` na server
- `ProcessMessage (string message)` – metodě jsou předávány zprávy, určené vyšším vrstvám. Metoda rozlišuje, zda jde o zprávu ve formátu MIME nebo příkaz protokolu IMAP. MIME zprávy rozbaluje a jejich obsah (objekty třídy `DataObjects`) předává poskytovateli dat.
- `SendData (DataObject o, boolean? update)` – odešle nový či pozměněný datový objekt serveru zabalený do zprávy MIME. Následně čeká na potvrzení o vložení dat do databáze. V případě, že je požadavek kladně vyřízen, objekt `o` je předán poskytovateli dat, který s ním příslušně naloží. Server tedy nemusí zasílat nově vložená data zpět klientovi. Parametr `update` může nabývat tří hodnot a určuje, zda jde o aktualizaci (`true`), nový objekt (`false`) či žádost o smazání objektu (`null`).

### 5.3.3.4 Třída `CommunicationLayer`

Tato třída realizuje fyzické spojení se serverem. Použitá technologie odpovídá té, která je použita u aplikačního serveru. Jde o asynchronní komunikaci prostřednictvím socketů.

Položky:

- `receiveBufferSize` – konstanta, určující velikost bufferu pro příjem dat. Je nastavena na 1024 bytů.
- `BasePort` – port, na kterém probíhá komunikace. Výchozí hodnota je 14241
- `Active` – booleovská hodnota, která ukazuje, zda je spojení se serverem k dispozici

Metody:

- `Connect()` – metoda je zavolána po spuštění aplikace, případně při požadavku na znovunavázání spojení. Otevře nový socket, jako protokol zvolí TCP. Na socketu zavolá metodu `BeginConnect`, které zadá IP adresu serveru, port a delegátem metodu `ConnectCallback`, která bude zavolána při úspěšném připojení.
- `ConnectCallback()` – je zavolána, akceptuje-li serverový socket příchozí spojení. Následně zašle serveru zprávu se jménem klienta a začne na socketu naslouchat.
- `ClientReceiveCallback()` – metoda je delegátem předána socketu, který ji zavolá vždy, když obdrží zprávu. Jde o hlavní metodu třídy, která rozlišuje typy zpráv. Ty komunikační předává vyšší vrstvě.
- `ConnectionTest()` – slouží k ověření dostupnosti serveru. Metoda se pokouší zaslat na server testovací zprávu. Vrací booleovskou hodnotu v závislosti na úspěšnosti odeslání zprávy.

- `DownloadStream(Stream stream, long length)` – metoda, která stáhne ze serveru datový proud délky `length` a ukládá jej do streamu `stream`.
- `SendStream(Stream stream, long length)` – zašle na server datový proud `stream` délky `length`
- `SendMessage(string message char type)` – metoda, sloužící k zaslání zpráv na server. Znak `type` specifikuje typ zprávy, řetězec `message` může obsahovat další parametry.
- `Send(string message)` – veřejně přístupná metoda, kterou využívají vyšší vrstvy ke komunikaci
- `OnSendMessage()` – metoda, volaná asynchronně vždy po odeslání zprávy. Ukončuje odesílání.
- `OnClosing()` – metoda je volána při ukončení programu, případně spojení. Pokusí se zavřít socket, na kterém probíhá komunikace.

### 5.3.4 Moduly

Každý z implementovaných modulů se specializuje na práci s daty jednoho typu. Pluginy jsou si ve výsledku velmi podobné, liší se funkcemi a uživatelským rozhraním, které odpovídá typu dat, na který je plugin zaměřen.

Základem každého modulu je třída, implementující rozhraní `IPlugin`. To je velmi jednoduché a neklade na implementaci téměř žádné požadavky. Je nutné do třídy zahrnout položky `Name`, `Description`, `Author` a `Version`, určující pořadí jméno modulu, popis, autora a verzi. Dále je nutné deklarovat (nikoli implementovat) metody `Initialize()` a `Dispose()`, které se volají při vytváření a rušení instance modulu. Každý plugin by měl mít také ikonu, která se zobrazí v hlavním menu aplikace a volitelně položku `Menu`, která se zařadí do lišty s menu v horní části okna aplikace. Důležitou součástí každého pluginu je prvek uživatelského rozhraní `MainInterface`, který se zobrazí v hlavním okně, bude-li plugin aktivní. Poslední položkou rozhraní je `IPluginHost Host`, které umožňuje pluginu využívat metody, implementované v jádru aplikace.

#### 5.3.4.1 Postup při implementaci nového modulu

Nejprve je nutné vytvořit třídu, která implementuje rozhraní `IPlugin`. Následně je potřeba vytvořit nový prvek uživatelského rozhraní, jehož instance se při inicializaci modulu přiřadí do proměnné `MainInterface`. Implementace dalších funkcí je zcela v režii autora pluginu. Součástí aplikace jsou moduly, které si vyžádají referenci na kolekci dat konkrétního typu a s těmito daty nějakým způsobem pracují. Například modul správy projektů po spuštění požádá o reference na kolekci projektů, úkolů a uživatelů. Kolekci projektů zobrazuje v grafickém rozhraní a ze zbylých kolekcí čerpá doplňující data například o úkolech, ze kterých se konkrétní projekt skládá nebo o uživateli, kteří se na projektu podílí. Dále využívá metod

`NewDataObject()`, `UpdateDataObject()` a `DeleteDataObject()`, které jsou přístupné proměnnou `Host`, k vytváření, aktualizaci a mazání projektů.

Implementace vlastního modulu tedy vyžaduje znalost rozhraní `IPlugin` a `IPluginHost`, kolekce `DataCollection` a třídy objektů `DataObjects`.

Po dokončení implementace je potřeba vytvořit dynamickou knihovnu (soubor `*.dll`) a umístit ji do adresáře s nainstalovaným programem.

### 5.3.4.2 Příklad implementovaného modulu

Implementace klientské části aplikace zahrnuje realizaci pěti základních modulů. Jedním z nich je plugin, sloužící ke správě projektů a úkolů, na kterých se uživatel podílí. Implementace tohoto modulu poslouží jako příklad pro vytváření dalších pluginů.

Základní třída, která modul reprezentuje, je nazvaná `ModuleProjects`. Jak je patrné z obrázku 8, implementuje tato třída rozhraní `IPlugin`. Pomocí její instance tedy s pluginem komunikuje jádro aplikace a naopak. `ModuleProjects` obsahuje položku `myHost`, která zpřístupňuje všechny metody a položky rozhraní `IPluginHost`.

Aby byly datové kolekce a služby aplikace přístupné z všech míst pluginu bez nutnosti mít referenci na rozhraní `IPluginHost`, zastřešuje třída `ModuleProjects` položky a metody vlastními metodami. Například položka `Projects` zpřístupňuje odkudkoli v modulu kolekci s projekty:

```
public DataCollection Projects
{
    get { return myHost.Projects(); }
}
```

Metoda `NewDataObject()` slouží k vložení nových dat do databáze. Odpovídající tabulka je zvolena na základě typu zaslaného objektu `o` (tuto problematiku řeší server). Návrátová hodnota značí, zda operace proběhla správně. Pokud ne, je jedním z argumentů řetězec `errorMessage`, který uživatele informuje o chybě. Plugin by měl uživateli tuto zprávu zprostředkovat.

```
public bool NewDataObject(DataObjects.DataObject o,
                          out string errorMessage)
{
    return myHost.NewDataObject(o, out errorMessage);
}
```

Dalšími třídami, které modul obsahuje, jsou `Tasks` a `Projects`. Z obrázku 8 je vidět, že obě dvě dědí od třídy `UserControl`, což je prvek uživatelského rozhraní. Tyto dvě třídy tedy implementují grafické součásti modulu. Třída `Projects` obsahuje prvky, určené k zobrazení projektů, na kterých uživatel pracuje, umožňuje je editovat a prohlížet jejich detaily. To vše s referencí na třídu `ModuleProjects`, díky které získá data o projektech, případně vloží nové projekty

do databáze. Podobně funguje třída `Tasks`, jen s tím rozdílem, že pracuje s úkoly konkrétního projektu.

Nyní lze ukázat, jak fungují pozorovatelné kolekce v praxi. Přehled projektů je zobrazen v tabulce jménem `DataGrid`. Tabulka předpokládá, že bude zobrazovat objekty typu `ProjectDataObject` a pro každý její sloupec je definovaný atribut projektu, který se v něm zobrazí. Tabulku pak lze naplnit jediným příkazem

```
DataGrid.DataContext = Projects;
```

kde `Projects` představuje kolekci projektů, získanou z hlavního programu.

Pro zobrazení přehledu úkolů, náležících do jednoho projektu, je potřeba aplikovat jiný postup. Nejprve se vytvoří prázdná lokální kolekce úkolů, nazvaná například `localTasks`. Dále je nutné implementovat metodu, která bude fungovat jako filtr. Její implementace může vypadat následovně:

```
public void Tasks_Changed(object sender, EventArgs e)
{
    TaskDataObject task = (TaskDataObject)sender;
    if (task.project == selectedProject)
    {
        localTasks.Add(tdo);
    }
}
```

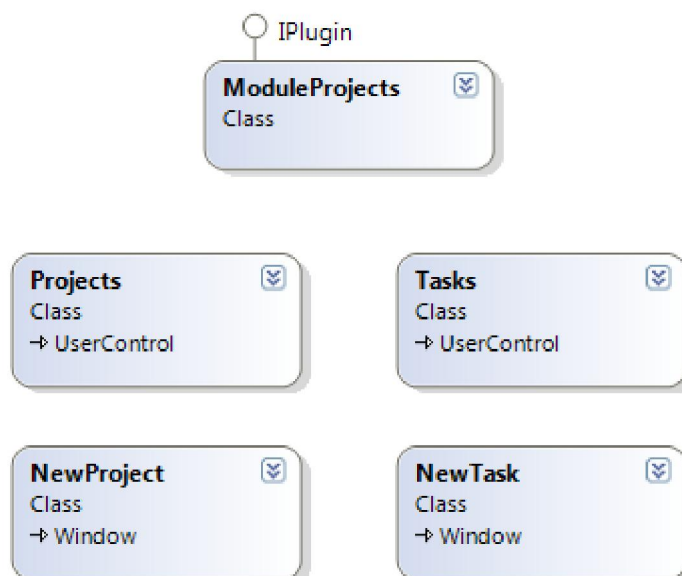
Tato metoda se přiřadí delegátovi hlavní kolekce s úkoly:

```
Tasks.Changed += Tasks_Changed;
```

a vždy, když bude do seznamu úkolů přidán nový úkol, zavolá se metoda `Tasks_Changed`, která ověří, zda má být úkol přidán do lokální kolekce. Tabulku s těmito úkoly stačí opět naplnit přiřazením lokální kolekce jejímu datovému kontextu. Delegát může obsahovat neomezené množství referencí na metody, každý plugin tedy může disponovat mnoha lokálními kolekcemi. Podrobněji je žádost o kolekci dat popsána v kapitole 5.3.5.1.

Třída `ModuleProjects`, která implementuje rozhraní `IPlugin`, musí obsahovat také referenci na jednu položku typu `UserControl`, která bude zobrazena v hlavním okně po tom, co bude modul inicializován. Právě z toho důvodu je `UserControl` společným předkem tříd `Projects` a `Tasks`.

Zbylé třídy `NewProject` a `NewTask` mají společného předka `Window`. Nejde tedy o elementární části grafického rozhraní, které je možné integrovat do existujícího okna, ale o samostatná okna, sloužící k zadání údajů o novém projektu či úkolu.



Obrázek 8 - Schéma tříd modulu Projects

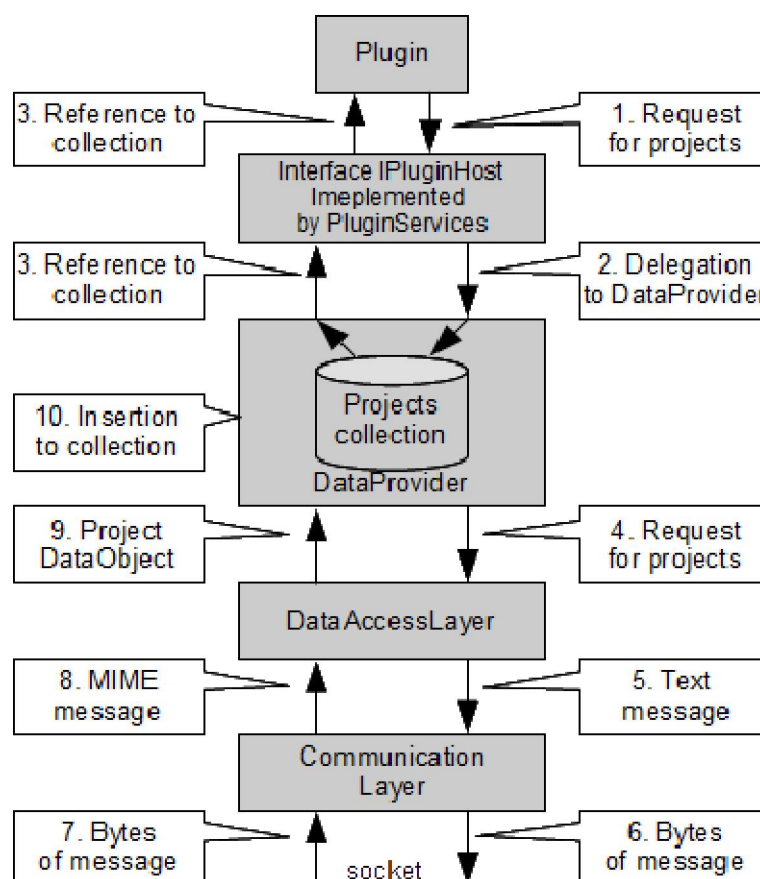
### 5.3.5 Zpracování dat

Na závěr jsou uvedeny tři příklady, popisující nejběžnější operace s daty, které program vykonává.

#### 5.3.5.1 Žádost modulu o kolekci dat

1. Modul žádá prostřednictvím metody `Projects()` rozhraní `IPluginHost` o kolekci objektů typu `ProjectDataObject`.
2. V jádru aplikace implementuje dané rozhraní třída `PluginServices`, která předá požadavek na kolekci projektů stejnojmenné metodě ve třídě `DataProvider`.
3. V `DataProvideru` je žádost delegována metodě `GetData()`. Mohou nastat dvě situace. V prvním případě `DataProvider` zatím kolekcí projektů nedisponuje. Vytvoří tedy novou kolekci objektů typu `ProjectDataObject` a zařadí ji do hašovací tabulky k ostatním kolekcím. V druhém případě již kolekce s projekty existuje a pravděpodobně všechny relevantní objekty projektů obsahuje. V takové situaci `DataProvider` nevytváří novou kolekci, ale vrátí referenci na existující.
4. Pokud kolekce neexistovala, zažádá `DataProvider` prostřednictvím reference na vrstvu přístupu k datům o zaslání žádosti o projekty na server. Jde o metodu `DataRequest()`.
5. Třída `DataAccessLayer` formuluje požadavek do textové zprávy a předá ji komunikační vrstvě k zaslání na server.

6. Komunikační vrstva textovou zprávu transformuje do seznamu bytů a předá je socketu k odeslání.
7. Modul prozatím disponuje referencí na kolekci dat, která může být prázdná. To je ovšem situace, se kterou musí implementace počítat. V optimálním případě začnou velmi brzy přicházet ze serveru data. Nejprve komunikační vrstvě ve formě přijatých bytů.
8. Komunikační vrstva byty interpretuje jako textový řetězec, ten identifikuje jako zprávu určenou vyšší vrstvě a předá jej třídě `DataAccessLayer`, konkrétně metodě `ProcessMessage()`.
9. Metoda rozezná, že jde o zprávu ve formátu MIME a dle parametrů zprávy (například dle typu obsahu) pozná, že příloha má textový charakter formátu `iCalendar`, tedy že jde o záznam o projektu. Stejným způsobem jako stejnojmenná vrstva na serveru vytvoří z obsahu objekt typu `ProjectDataObject` a předá jej metodě `InsertDataObject()` vrstvy `DataProvider`.
10. `InsertDataObject()` funguje nezávisle na tom, že před okamžikem stejná třída iniciovala zaslání dat ze serveru. Podívá se, jaký typ má objekt, který dostala parametrem a dle tohoto typu zvolí kolekci, do které objekt zařadí. Může se stát, že se v kolekci objekt s daným identifikátorem již nachází. V takové situaci implementace počítá s tím, že objekt, přijatý ze serveru, je aktuální a původní objekt v kolekci jím nahradí.



Obrázek 9 - Zpracování žádosti o projekty

### 5.3.5.2 Vytvoření lokální kolekce objektů jednoho typu, které splňují danou podmínku

1. Modul požaduje kolekci všech uživatelů, pracujících ve skupině číslo 1. Nejprve požádá prostřednictvím rozhraní `IPluginHost` o referenci na kolekci uživatelů. Z uvedené charakteristiky vyplývá, že taková kolekce obsahuje všechny uživatele ze všech pracovních skupin, ve kterých pracuje přihlášený uživatel. Není to tedy zatím to, co modul očekává. Veškeré operace, spojené s vracením reference na kolekci uživatelů, jsou stejné jako v předchozím případě.
2. Modul nyní disponuje kolekcí všech kontaktů, ke kterým má uživatel přístup. Jelikož jde o pozorovatelnou kolekci, dokáže tato datová struktura zaznamenat změnu a zareagovat na ni. K tomuto účelu disponuje každá pozorovatelná kolekce delegátem, do kterého je možné přiřadit metody, které se v případě změny zavolají. Modul si tedy vytvoří novou lokální kolekci, určenou pro objekty typu

`UserDataObject` a k původní kolekci přiřadí metodu, která vždy při změně aktualizuje také lokální kolekci. Taková metoda tedy představuje filtr – je implementována tak, že pokud bude do kolekce všech kontaktů přidán nový kontakt, který je zařazen do pracovní skupiny číslo 1, bude vložen také do lokální kolekce pluginu. Metoda je tedy součástí pluginu a hlavní kolekci s uživateli je předána pouze reference na ni. Stejně tak pokud bude z hlavní kolekce odstraněn uživatel, který pracuje ve skupině číslo 3, bude metoda vědět, že takový uživatel se v lokální kolekci nenachází a změnu bude ignorovat.

### 5.3.5.3 Vytvoření nového objektu

1. Modul vytvoří nový objekt typu `AppointmentDataObject` a předá jej metodě `NewDataObject()` rozhraní `IPluginHost`, tedy třídě `PluginServices`, která rozhraní implementuje.
2. Třída `PluginServices` deleguje objekt vrstvě `DataAccessLayer`, konkrétně metodě `SendData()`, která objekt serializuje do podoby textového řetězce ve formátu `iCalendar`, přiloží jej ke zprávě ve formátu `MIME` a jako typ obsahu uvede `text/calendar`. Potom zprávu jako text předá komunikační vrstvě, která jej odešle serveru.
3. Server zprávu zpracuje dle postupu uvedeného v kapitole 5.2.3.2 a zašle klientovi výsledek operace v textové podobě.
4. U klienta zpracuje potvrzení stejná třída, jako ta, která data odesílala – tedy `DataAccessLayer`. Tato vrstva si celou dobu drží referenci na nově vytvářený objekt. Potvrdí-li server vložení objektu do databáze, předá třída vrstvě `DataProvider` tento objekt jako nově přijatý ze serveru a `DataProvider` jej zařadí do příslušné kolekce.

## 6 Závěr

Hlavním cílem této práce bylo navrhnout a implementovat nenáročnou a snadno použitelnou groupware aplikaci s tlustým klientem, která zkombinuje jednoduchost webových programů s komfortem uživatelského rozhraní, které nabízí tlustý klient.

Návrh a implementace vycházejí z kritérií, uvedených v sekci 3.1. Aplikace je modulární, implementace jednotlivých pluginů reflektuje požadavek jednoduchosti a přehlednosti a struktura pracovních skupin, projektů a úkolů odpovídá nárokům běžně kladeným na groupware aplikace.

Pluginy podporují práci se všemi požadovanými základními typy dat, jakými jsou projekty, úkoly či kontakty. Funkce vybraných pluginů jsou nahraditelné

aplikacemi třetích stran. To se týká především modulů, pracujících s kalendářovými položkami a kontakty. Díky formátům iCalendar a vCard, které byly zvoleny k reprezentaci kontaktů a událostí v průběhu přenosu dat mezi částmi programu, lze tomuto požadavku vyhovět velmi snadno, neboť se jedná o otevřené formáty, hojně implementované mnoha různými aplikacemi podobného zaměření.

Požadavek modularity nejvýrazněji ovlivnil podobu klientské části aplikace. Výsledkem je, že se lze pro každý plugin nezávisle rozhodnout, zda jej používat či nikoli. Pluginy se vzájemně neovlivňují, jejich funkce tedy zůstávají stejné bez ohledu na to, zda je zapnutý nějaký jiný modul. Popsané rozhraní pro komunikaci s jádrem programu umožňuje kdykoli rozšířit aplikaci o další plugin.

Poněkud problematickou se ukázala kompatibilita s Kolab serverem. Příprava aplikace na kompatibilitu se serverem Kolab ovlivnila zejména způsob komunikace a formát přenášených dat. Pro aplikaci jako takovou je tento požadavek velmi přínosný. Místo návrhu vlastního přenosového protokolu jsou data přenášena jako přílohy MIME zpráv, čímž se značně zlepšila rozšiřitelnost aplikace. Ačkoliv návrh a implementace přenosových vrstev a volba formátu dat odpovídají požadavkům Kolab serveru, aplikace ve stávající verzi plně kompatibilní není. Důvodem je, že pro dosažení kompatibility by bylo nutné slevit z některých požadavků na hierarchii projektů, úkolů a pracovních skupin a také neimplementovat sdílení souborů.

Výsledkem tedy je, že aplikace je připravená pro komunikaci s Kolab serverem bez nutnosti měnit implementaci komunikačních vrstev, ovšem hierarchie některých dat, se kterými pracuje, Kolabu neodpovídá. K docílení úplné kompatibility by bylo nutné změnit implementaci některých modulů, konkrétně projektů a správy pracovních skupin, nebo konfigurovat aplikaci tak, aby bylo možné přepínat mezi standardním režimem, v němž by klient komunikoval s nativním serverem, a omezeným režimem, ve kterém by komunikace probíhala s Kolabem. Tato varianta nebyla při návrhu aplikace zásadním požadavkem a zůstala proto jako možnost rozšíření aplikace do budoucna.

Z hlediska analýzy groupware softwaru se aplikace řadí do skupiny CMT. Nabízí elementární funkce typické pro groupware a neposkytuje komplexní prostředí pro práci uživatelů. Rozsah funkcí programu je tedy velmi podobný většině groupware aplikací založených na internetových technologiích, které jsou uvedeny ve druhé kapitole. Přesto je však implementovaná aplikace odlišná, neboť je možné ji používat bez přístupu k Internetu a mít data potřebná k práci neustále k dispozici, což žádný webový groupware program již z principu nenabízí.

Je zřejmé, že s Microsoft Exchange Serverem či programem IBM Lotus Notes and Domino nelze výsledný software příliš srovnávat a program ani neměl takové ambice. Výsledná aplikace je naopak velmi blízký unixový Kontakt. Podobá se jí nejen základními koncepty, jakými jsou modularita, funkce CMT programů či kompatibilita s Kolabem, ale také funkcemi a základními implementovanými moduly. Aplikace tedy do jisté míry nabízí alternativu ke Kontaktu pro systémy na platformě Windows.

## 7 Literatura

[1] Coleman, D. and R. Khanna (editors). Groupware: Technology and Applications. Englewood Cliffs, NJ: Prentice-Hall PTR, 1995.

[2] Udell, Jon: Practical internet groupware. Sebastopol, Calif. : O'Reilly & Associates, 1999.

[3] Dokumentace serveru Kolab:  
URL: <http://www.kolab.org/doc/kolabformat-2.0rc7-html/index.html>

[4] Microsoft (služby Exchange Serveru):  
URL: <http://www.microsoft.com/cze/servers/exchange/default.mspx>

[5] Web IBM (informace o Lotus Notes and Domino):  
URL: <http://www-01.ibm.com/software/lotus/products/notes/>

[6] Groupware, CRM na server root.cz  
<http://www.root.cz/groupware-crm/>

[7] Web DotProject (informace, dokumentace, demo)  
URL: [www.dotproject.net/](http://www.dotproject.net/)

[8] Web MoreGroupWare + dokumentace (tamtéž)  
URL: <http://www.moregroupware.de/>

[9] Web PHP Project (informace, dokumentace, demo)  
URL: <http://www.phprojekt.com/>

[10] Web PHP Groupware (dokumentace, demo)  
URL: <http://docs.phpgroupware.org/>

# A. Uživatelská příručka

## Instalace programu

Klientská aplikace pro svůj běh vyžaduje přístup k aplikačnímu serveru. Pokud jej dosud nemáte zprovozněný, nainstalujte a zkonfigurujte jej dle přílohy B níže. Pro instalaci klientského programu z příloženého CD proveďte následující kroky:

- Vložte do mechaniky přiložený kompaktní disk.
- Zobrazte obsah disku a otevřete složku `install`.
- Otevřete složku `client`.
- Spusťte soubor `setup.exe`.
- Program vyžaduje ke správnému fungování rozhraní Microsoft .NET ve verzi 3.5 nebo vyšší. Pokud jej nemáte nainstalované, instalace Vám bude nabídnuta. Klepnutím na `Accept` ji potvrdíte, rozhraní se automaticky stáhne z internetu a nainstaluje. Není-li připojení k internetu k dispozici, je nutné nainstalovat rozhraní jiným způsobem, jinak není možné program používat.
- Je-li .NET nainstalovaný, automaticky se zahájí instalace vlastní aplikace. V prvním kroku je nutné potvrdit, že chcete software skutečně nainstalovat. Pokud souhlasíte, klepněte na `Next`.



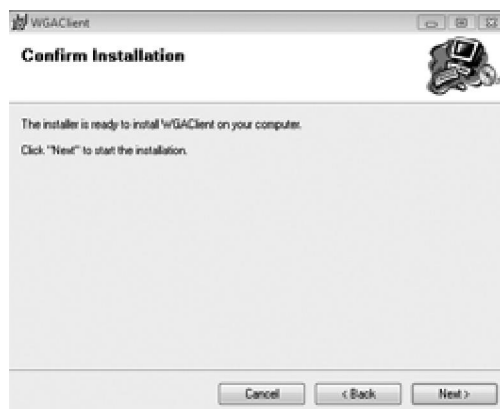
Obrázek 10 - Potvrzení instalace

- Nyní je možné změnit složku, do které se program nainstaluje. Kliknutím na tlačítko `Browse` můžete vybrat jiné umístění, než které je přednastavené. Pokračujte tlačítkem `Next`.



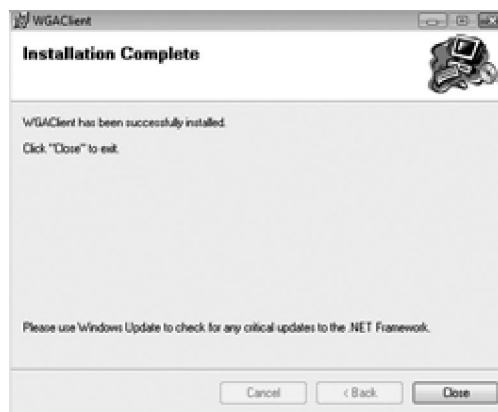
**Obrázek 11 - Výběr složky, do které se program nainstaluje**

- Následující okno pouze informuje o tom, že je možné instalaci zahájit. Potvrďte kliknutím na `Next`.



**Obrázek 12 - Potvrzení instalace**

- Nyní proběhne instalace programu. V závislosti na rychlosti počítače může trvat několik sekund, případně minut. Po skončení instalace zavřete instalátor kliknutím na `Close`.



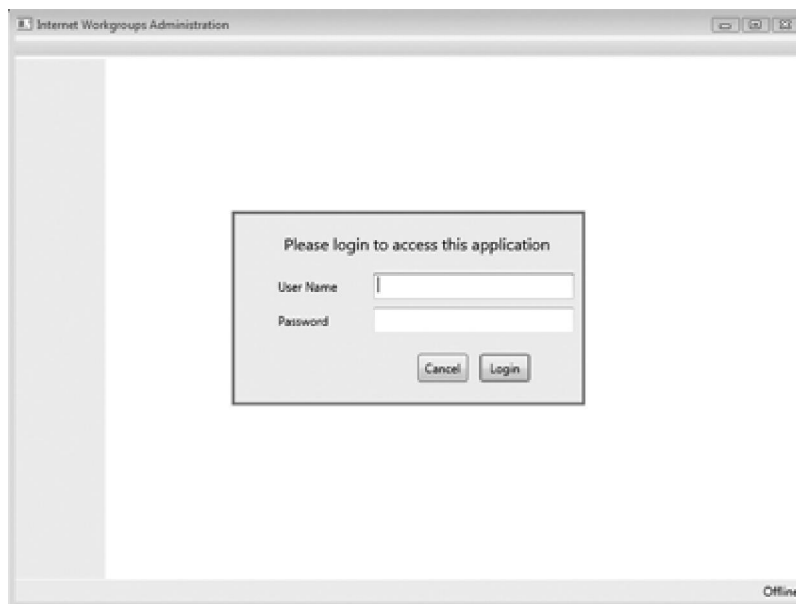
**Obrázek 13 - Dokončení instalace**

- Aplikaci spustíte poklepnutím na zástupce jménem `WGA Client`, kterého najdete na pracovní ploše, případně v nabídce Start.

Poznámka: Aby se aplikace mohla připojit k serveru, je nutné znát jeho IP adresu a port. Více informací o změně těchto údajů se nachází v konfigurační příručce v kapitole Konfigurace klientské části aplikace. Jako výchozí je nastavena IP adresa 127.0.0.1 a port číslo 14241. Pokud se Vám nepodaří přihlásit se do aplikace (viz další kapitola), je možné, že jsou údaje špatně nastavené. V takovém případě je nezbytné kontaktovat správce aplikačního serveru, který zná správné hodnoty portu a adresy IP.

## Přihlášení

Po spuštění aplikace se zobrazí okno, požadující vložení údajů nutných k přihlášení. Zadejte své uživatelské jméno a heslo a klepnutím na tlačítko `Login` se přihlaste. Jde-li o první přihlášení k programu na Vašem počítači, je nutné, abyste byli připojeni k internetu, případně k síti, ve které se nachází aplikační server. V opačném případě program nedokáže ověřit Vaši identitu. Při dalších přihlášeních již není nutné být online. V případě, že se přihlásíte z jiného počítače a změňte si heslo, při prvním online přihlášení z Vašeho počítače bude požadováno nové heslo, stejně jako při dalších offline přístupech k programu.

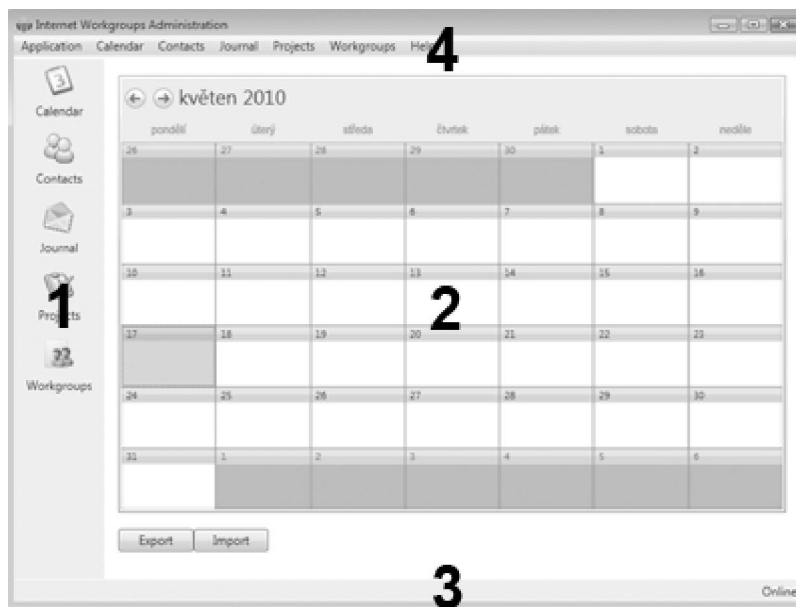


Obrázek 14 - Přihlašovací okno programu

## Hlavní nabídka

Pokud proběhne přihlášení úspěšně, zobrazí se hlavní okno aplikace. Je možné jej rozdělit na čtyři hlavní části, jak je vidět na obrázku 15.

- Vlevo je umístěn sloupec s hlavní nabídkou aplikace. Slouží k přepínání mezi různými funkcemi programu, díky nimž je možné zobrazit kalendář, seznam kontaktů a další funkce, které program nabízí. Na obrázku 15 se nabídka nachází pod číslem 1.
- Největší část okna zabírá prostor, který pro zobrazování údajů využívají jednotlivé pluginy. Na obrázku 15 je označený číslem 2. Obsah této části závisí vždy na modulu, který je zrovna používán, v tomto prostoru může tedy být zobrazen například kalendář.
- Stavový řádek je tvořen spodní lištou, na obrázku 15 je označený číslem 3. Vpravo na této liště je možné sledovat, zda je program připojený k serveru či nikoli.
- Číslo 4 na obrázku 15 označuje hlavní menu, které může mít proměnlivý počet položek v závislosti na tom, jaké pluginy jsou aktivované. Položky *Application* a *Help* jsou však přítomné vždy.



Obrázek 15 - Části hlavního okna

## Nastavení

Standardní distribuce aplikace obsahuje pět základních modulů, které je možné využívat: Kalendář, kontakty, žurnál, správu projektů a správu pracovních skupin. Ve výchozím nastavení jsou všechny tyto moduly aktivní, po prvním přihlášení se proto jejich ikony zobrazí v hlavní nabídce.

Pokud si nepřejete některý z modulů používat, je možné jej zcela deaktivovat. V hlavním menu klepněte na `Application` a dále na `Options`. Zobrazí se tabulka se třemi sloupci: V prvním sloupečku je jméno modulu, ve druhém zaškrťovací políčko a ve třetím stručný popis, k čemu modul slouží. Moduly, které jsou aktivní, mají v prostředním sloupci políčko zaškrtnuté. Najděte tedy řádek s modulem, který nechcete nadále používat a odškrtněte políčko v prostředním sloupci. Potvrďte tlačítkem `Apply`. Změna se projeví po restartu aplikace.

### Available / Preferred plugins

Name	Allowed	Description
Calendar	<input checked="" type="checkbox"/>	Displays calendar.
Contacts	<input type="checkbox"/>	Shows contacts.
Journal	<input checked="" type="checkbox"/>	Shows journal.
Projects	<input type="checkbox"/>	Shows workgroups, projects and users in a table.
Workgroups	<input checked="" type="checkbox"/>	Workgroups administration.

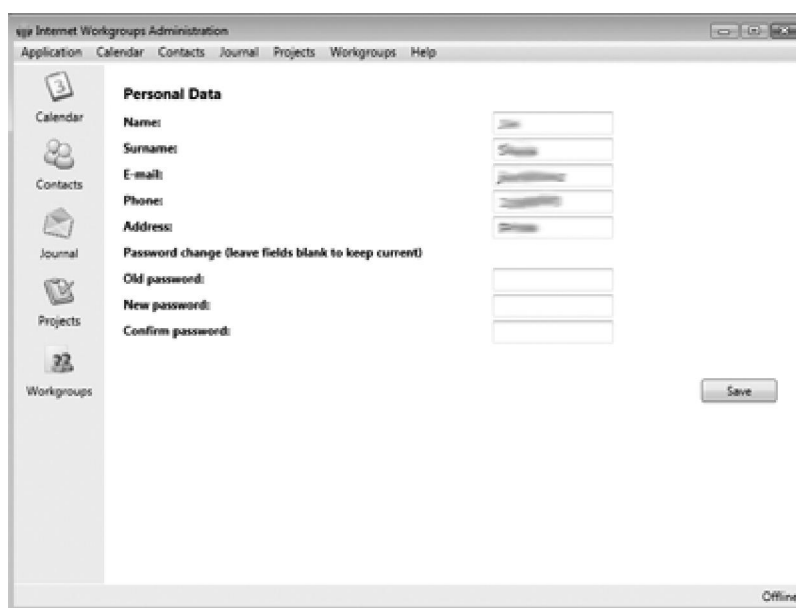
Obrázek 16 - Aktivace / deaktivace modulů

Pokud si přejete nějaký modul znovu aktivovat, na stejném místě zaškrtněte políčko u tohoto modulu. V případě, že se jedná o zcela nový modul, který s programem není dodáváný, nahrajte soubor (případně soubory) s tímto modulem do složky, kde je aplikace nainstalována a program restartujte. Je-li modul s aplikací kompatibilní, nabídne se jeho aktivace v nabídce Options.

## Změna uživatelských údajů

Změnit jméno, příjmení, e-mailovou adresu, telefon či heslo je možné prostřednictvím nabídky Personal Settings, která se nachází v hlavním menu Application.

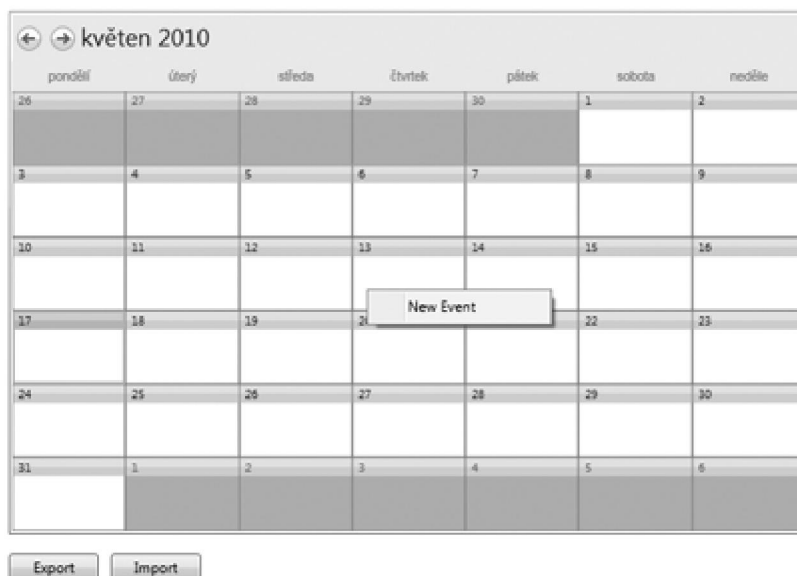
Prvních několik polí obsahuje osobní údaje, zbylá tři políčka slouží ke změně hesla. Zůstanou-li nevyplněná, heslo zůstane původní. V opačném případě je nutné vyplnit stávající heslo, do zbylých dvou políček napsat nové heslo a kliknutím na Save potvrdit změny.



Obrázek 17 - Editace osobních údajů

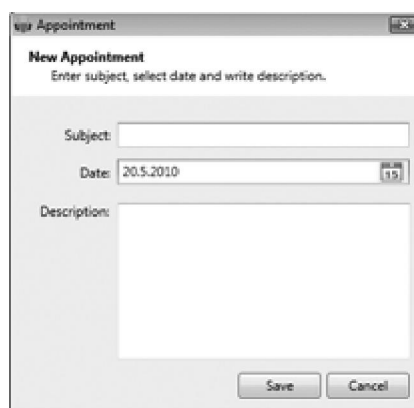
## Kalendář

Kalendář zobrazuje důležité události, týkající se přihlášeného uživatele. Označuje termíny, určené počátečními a koncovými daty projektů a úkolů, na kterých se uživatel podílí. Tyto události mají informativní charakter a není možné je měnit. Slouží k upozornění, že se blíží termín, do kterého má být úkol splněný. Kromě toho jsou viditelná také data úkolů, které řeší ostatní uživatelé, kteří se na projektu podílí.



**Obrázek 18 - Kalendář s kontextovým menu, určeným k vytvoření nové události**

Je možné vytvářet vlastní, soukromé události, které jsou viditelné pouze ve Vašem kalendáři. Klikněte pravým tlačítkem do čtverce, který symbolizuje den, odpovídající datu nové události. Zobrazí se kontextová nabídka s položkou **New Event**, na kterou klepnete. Otevře se nové okno, obsahující tři pole:



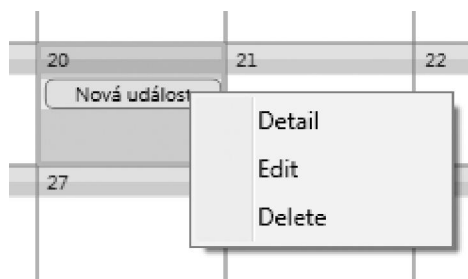
**Obrázek 19 - Okno pro vytvoření nové události**

Do prvního pole s názvem **Summary** vepište stručný název události. Ten bude zobrazen v kalendáři. Do řádku **Date** můžete vyplnit datum události, neshoduje-li se s datem, které je předvyplněné. Nakonec do pole **Description** vepište podrobný popis události. Tlačítkem **Save** událost uložte. Událost se zobrazí v kalendáři u patřičného data:



Obrázek 20 - Událost zobrazená v kalendáři

Chcete-li si prohlédnout detail události, klepněte na ni v kalendáři pravým tlačítkem a zvolte možnost `Detail`.



Obrázek 21 - Kontextové menu položky kalendáře

Události, které vytvoříte, je možné upravovat a mazat. Obě možnosti lze najít v kontextovém menu, které se objeví po kliknutí na název události pravým tlačítkem. `Edit` zobrazí okno, v němž je možné událost upravit a `Delete` událost smaže.

V dolní části okna pod kalendářem se nacházejí dvě tlačítka. Prvním tlačítkem s nápisem `Export` je možné veškeré události uložit do souboru ve formátu `*.ics`. Jde o formát `iCalendar`, který je používán mnoha jinými programy, například aplikací `Windows Calendar`. Události je tedy možné tímto způsobem zálohovat či importovat do jiného programu.

Druhé tlačítko `Import` provádí opačný proces. Otevře okno, které slouží k vybrání existujícího souboru ve formátu `iCalendar`. Události, které jsou v tomto souboru uloženy, budou vloženy do kalendáře. Původní položky, které se v kalendáři nacházely, budou nahrazeny, pokud mají stejné datum a název. Zbylé zůstanou zachovány. Pomocí importu a exportu tak lze jednoduše provádět zálohu dat.

## Kontakty

Modul `Kontakty` představuje jednoduchý nástroj, určený k zobrazování kontaktních informací o uživatelích, kteří pracují ve stejných skupinách jako přihlášený uživatel.

Nejpodstatnější část obrazovky modulu zabírá seznam uživatelů. Jsou zobrazena jejich celá jména. Pokud je přihlášený uživatel zařazený do více pracovních skupin, nachází se v seznamu všichni uživatelé z těchto skupin. Kliknutím na jméno nějakého uživatele se rozvine řádek s detaily. Ten obsahuje e-mailovou adresu, přihlašovací jméno do aplikace (je v rámci aplikace jedinečné), adresu a telefon. Více informací o uživateli dostupných není.

Nad seznamem kontaktů vlevo se nachází rozbalovací seznam, označený textem `Select workgroup`. Tím je možné vybrat konkrétní pracovní skupinu. V seznamu kontaktů se pak zobrazí pouze uživatelé, kteří náleží do vybrané skupiny.

Na stejné úrovni jako filtr pracovních skupin, ovšem vpravo, je textové pole, určené k vyhledávání. Zadaný text se hledá v příjmeních a křestních jménech uživatelů.

Name
Jan Marecek
Jiri Sedlak
Alena Sedlackova
Dusan Micica
Tomas Pop
Marcel Tlustos
Marketa Florianova
Petra Adamcova
Jana Havlova

**Obrázek 22 - Kontakty - nahoře je patrný filtr pracovních skupin a pole pro vyhledávání**

Tlačítkem `Reset` se seznam kontaktů vrátí do výchozího stavu.

Pod seznamem uživatelů jsou situována dvě tlačítka. Tlačítkem `Export Contacts` lze uložit všechny kontakty, které se v seznamu nachází, do textového souboru ve formátu `*.csv` (hodnoty oddělené čárkou). Takový soubor lze mimo jiné otevřít například v programu MS Excel.

Druhým tlačítkem ve spodní části okna (`Generate vCard`) lze vytvořit z předem označeného kontaktu soubor ve formátu `vCard`. Nutné je, aby v seznamu kontaktů byl označen nějaký uživatel před tím, než na tlačítko kliknete. Import vlastních kontaktů možný není.

## Žurnál

Modul lze chápat jako komunikační prostředek mezi členy jedné pracovní skupiny. Hlavní okno modulu je rozděleno na dvě části. V horní polovině se zobrazují příspěvky uživatelů. Zobrazují se chronologicky dle času, kdy byly vloženy. Každý příspěvek je označen titulkem, jménem uživatele, který je jeho autorem a datem, kdy byl publikován. Pod těmito údaji se nachází vlastní text příspěvku.

Příspěvky je možné upravovat či mazat. Operace může provádět buď uživatel, který je autorem příspěvku nebo vedoucí pracovní skupiny. Editaci je možné provést kliknutím na tlačítko `Edit` v horní části příspěvku, smazání tlačítkem `Delete`, které se nachází hned vedle.

**Journal**

Select workgroup:

Nový příspěvek	<b>Jirí Sedlak</b>	17.5.2010	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
Zkouška zurnálu.				
Druhý příspěvek	<b>Jirí Sedlak</b>	17.5.2010	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
Druhý příspěvek do zurnálu od stejného uživatele.				

Title:

Obrázek 23 - Žurnál - v centrální části se zobrazují vložené příspěvky, spodní slouží k vkládání nových

Dolní část okna slouží k psaní příspěvků. Obsahuje řádek `Title`, sloužící k vepsání titulku příspěvku a pod ním velké textové pole, určené k formulaci vlastního textu. Tlačítkem `Submit` se příspěvek publikuje.

V horní části okna je seznam, který umožňuje přepínat pracovní skupiny, ve kterých přihlášený uživatel pracuje. Příspěvky jsou totiž publikovány pouze v rámci jedné skupiny.

## Projekty

Plugin Projekty je určen ke správě projektů a úkolů, umožňuje vytvářet nové projekty, nahrávat k projektům soubory, vpisovat k nim poznámky a jiné.

Hlavní okno sestává z několika částí. V dolní části se nachází tabulka, v které jsou zobrazeny všechny projekty, na kterých přihlášený uživatel pracuje. Kliknutím na některý řádek tabulky se v horní části zobrazí detailní informace o vybraném projektu. Konkrétně jde o název projektu, popis, jméno vedoucího, termín začátku, termín dokončení, pracovní skupina, která projekt řeší a procentuální vyjádření toho, jak projekt pokračuje. V pravé části se nachází dvě tabulky. Ta horní zobrazuje poznámky, připisované k projektu. Hned pod ní je políčko, kam je možné text poznámky vyplnit a kliknutím na **New Note** poznámku vložit. Spodní tabulka obsahuje seznam souborů, které jsou s projektem asociované. Libovolný ze souborů je možné si stáhnout kliknutím na **Download**.

**Projects** New Project

**Name:** Web Zeleny Group

**Description:** Navrh, graficky design, realizace a nasazeni webové prezentace firmy Zeleny Group, spol. s.r.o.

**Leader:** Jiri Sedlak

**Date of Start:** 30.6.2009

**End date:** 28.10.2009

**Finished (%):** 96

**Workgroup:** Atech - Webove prezentace

**Notes and Comments**

17.5.2010 poznámka Delete

17.5.2010 aktualizujte stav!! Delete

New Note

**Associated Files**

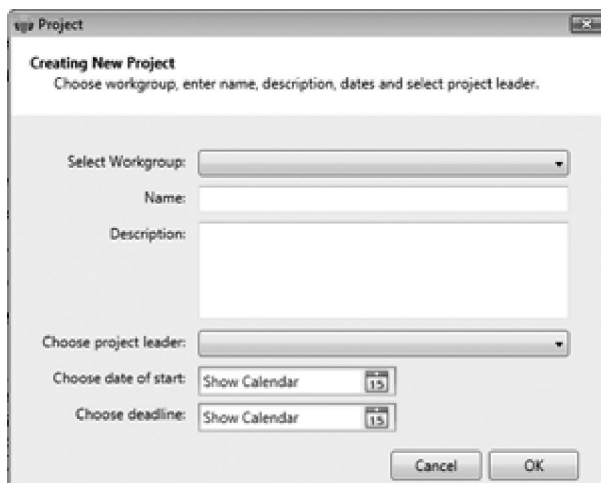
Name	Size	Download	Delete
DataObjects.pdb	71,5 KB	Download	Delete
Contacts.dll	21,5 KB	Download	Delete

Upload File

Project	Start Date	Deadline	% Finished	Tasks	Edit	Delete
IS MC Praha 10	3/22/2009 12:00:00 AM	8/15/2009 12:00:00 AM	100	Tasks	Edit	Delete
Web MU Slany	11/15/2009 12:00:00 AM	1/31/2010 12:00:00 AM	100	Tasks	Edit	Delete
Web Zeleny Group	6/30/2009 12:00:00 AM	10/28/2009 12:00:00 AM	96	Tasks	Edit	Delete

**Obrázek 24 - Projekty - dolní část obsahuje seznam projektů, centrální část vlevo details, v pravé části jsou tabulky s poznámkami a soubory**

Správce pracovní skupiny může vytvářet nové projekty. Okno s příslušnou nabídkou se zobrazí po kliknutí na tlačítko **New Project** v horní části obrazovky. Následně je třeba vyplnit jméno, popis, data počátku a ukončení a vybrat vedoucího projektu. Klepnutím na **Save** dojde k vytvoření projektu.



Obrázek 25 - Okno pro vytvoření nového projektu

Vedoucí projektu je oprávněn k projektu nahrávat soubory, které se zobrazí ve zmíněné tabulce. Je tak možné učinit kliknutím na Upload File pod tabulkou se soubory.

Právo smazat či upravit projekt má pouze vedoucí pracovní skupiny, ve které je projekt řešen.

V tabulce s projekty je u každého projektu tlačítko Tasks. Tím je možné se přesunout na obrazovku, ukazující úkoly, ze kterých se vybraný projekt skládá. Okno vypadá téměř stejně, jako to s projekty:

IS MC Praha 10 Back

Task Name: Testovani

Description:

User Name: Jan Marecek

Date of Start: 5.7.2009

Deadline: 5.8.2009

Finished: 100

Depends on: Priprava, shaneni podkladu  
Navrh  
Implementace

Notes and Comments

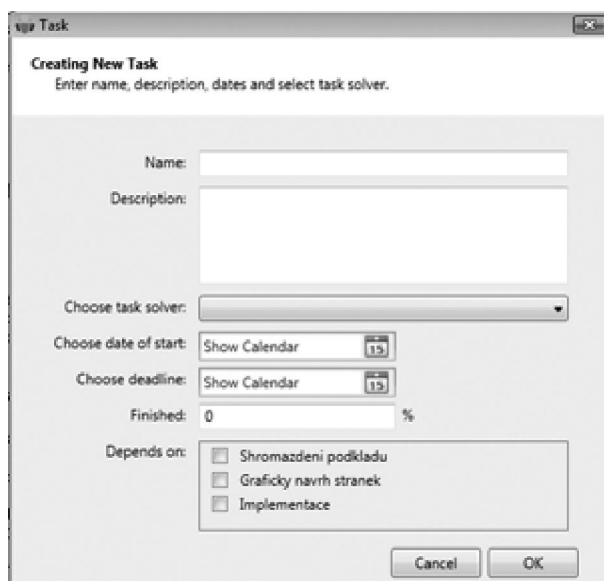
Associated Files

Task	Start Date	Deadline	% Finished			
Priprava, shaneni podkladu	3/22/2009 12:00:00 AM	4/14/2009 12:00:00 AM	100	Edit	Delete	
Navrh	5/1/2009 12:00:00 AM	6/1/2009 12:00:00 AM	100	Edit	Delete	
Implementace	6/14/2009 12:00:00 AM	8/2/2009 12:00:00 AM	100	Edit	Delete	
Testovani	7/5/2009 12:00:00 AM	8/5/2009 12:00:00 AM	100	Edit	Delete	

Obrázek 26 - Detail projektu

V dolní části je tabulka s jednotlivými úkoly, přičemž každý úkol je možné upravit nebo smazat. Učinit tak může pouze správce projektu. Kliknutím na libovolný z úkolů se v horní polovině okna zobrazí detaily – název úkolu, popis, řešitel, důležitá data a procento dokončení úkolu. Vpravo jsou pak stejné tabulky jako u přehledu projektů – tabulka s poznámkami a tabulka se soubory. Soubory

může tentokrát nahrávat také řešitel úkolu, a to kliknutím na tlačítko Upload File. V horní části se nachází tlačítko New Task, kterým může správce projektu vytvořit nový úkol.



Obrázek 27 - Vytvoření nového úkolu

## Pracovní skupiny

Nástroj Pracovní skupiny slouží běžnému uživateli pouze k zobrazení přehledu uživatelů, kteří s ním spolupracují, rozdělení uživatelů do skupin a do jednotlivých projektů, které skupina řeší. Komponenta nabízí stromový pohled na hierarchii pracovních skupin a uživatelů. Na nejvyšší úrovni se nachází jednotlivé pracovní skupiny. Každou skupinu je možné rozbalit – objeví se dvě složky. První se jmenuje All Users a obsahuje všechny uživatele, kteří do skupiny patří. Uživatelé, kteří mají vedle jména zobrazenou žlutou korunku, jsou vedoucími rozbalené skupiny. Ti s červenou korunkou jsou administrátoři v rámci společnosti. Druhá složka se jmenuje Users in Projects a jejími podsložkami jsou všechny projekty, které pracovní skupina řeší. Rozbalením jednotlivých projektů se zobrazí seznam uživatelů, kteří na daném projektu pracují.

## Workgroups administration



**Obrázek 28 - Správa pracovních skupin – nejvyšší úroveň obsahuje jednotlivé pracovní skupiny, nižší úroveň je dělena na složku se všemi uživateli a složku s projekty. Ve složce s projekty jsou vypsané projekty s uživateli, kteří se na nich podílí.**

Správce pracovní skupiny může uživatele v rámci své skupiny přiřazovat k projektům nebo je z nich odebírat. To je proveditelné kliknutím pravým tlačítkem na jméno uživatele a dále volbou `Copy user to project` pro přiřazení k projektu či `Remove user from project` k odebrání.



**Obrázek 29 - Jedno z kontextových menu u administrace skupin**

Administrátor může v rámci jedné společnosti vytvářet celé pracovní skupiny (tlačítko `New Workgroup` ve spodní části okna), nové uživatelské účty (tlačítko `New User` ve spodní části okna), přiřazovat uživatele do skupin nebo je z nich odebírat, stanovovat uživatele vedoucími pracovní skupiny, případně administrátory. Všechny operace jsou přístupné pomocí kontextového menu, které se zobrazí po kliknutí na jméno uživatele pravým tlačítkem.



Obrázek 30 - Vytváření nové pracovní skupiny



Obrázek 31 - Vytváření nového uživatelského účtu

## Příklad administrace pracovní skupiny

Tento příklad ukazuje, jakým způsobem může probíhat správa jedné pracovní skupiny, včetně vytváření projektů a přiřazování úkolů uživatelům. Předpokládá se tedy, že přihlášený uživatel je správcem pracovní skupiny.

1. Spustíte aplikaci a přihlaste se svým uživatelským jménem a heslem.
2. Kliknutím na *Workgroups* se přesuňte do části, určené ke správě pracovních skupin.
3. Nyní si můžete prohlédnout hierarchii uživatelů v pracovních skupinách, ve kterých jste zařazen. Po rozbalení pracovní skupiny a následně složky *All Users* se ukáže přehled všech uživatelů, kteří ve skupině pracují. Ti, kteří mají u jména žlutou korunku, jsou vedoucími skupiny.



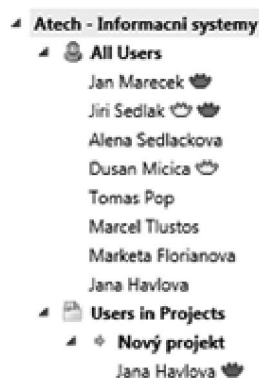
Obrázek 32 - Hierarchie uživatelů ve skupině

4. V hlavní nabídce klikněte na **Projects** a následně na tlačítko **New Project**, které se nachází v pravém horním rohu okna.
5. Zobrazí se okno stejné, jako na obrázku 25. Vyberte pracovní skupinu, která bude projekt řešit. Dále vyplňte název projektu, popis a zvolte vedoucího projektu. Jako vedoucího můžete zvolit libovolného uživatele, který pracuje ve skupině, kterou jste pro projekt vybrali. Jiné uživatele Vám program nenabídne. Nakonec vyplňte počáteční a koncové datum a potvrďte stisknutím **OK**.
6. Nový projekt se přidá do seznamu v dolní části okna.

Project	Start Date	Deadline	% Finished			
Nový projekt	4/28/2010 12:00:00 AM	5/22/2010 12:00:00 AM	0	Tasks	Edit	Delete
Web MU Slany	11/15/2009 12:00:00 AM	1/31/2010 12:00:00 AM	100	Tasks	Edit	Delete
Web Zeleny Group	6/30/2009 12:00:00 AM	10/28/2009 12:00:00 AM	95	Tasks	Edit	Delete

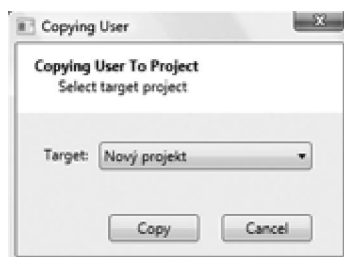
Obrázek 33 - Seznam projektů s nově vytvořeným projektem

7. Přesuňte se zpět do části **Workgroups**. Nový projekt se přidal také do hierarchie uživatelů v dané pracovní skupině, konkrétně do složky **Users in Projects**. Jediným uživatelem, který je do projektu přiřazen, je vedoucí, kterého jste zvolili při vytváření. Vedoucí projektu je označen modrou korunkou, uvedenou u jména.



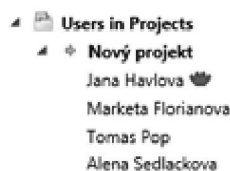
Obrázek 34 - Hierarchie uživatelů skupiny po přidání nového projektu

8. Úkolem vedoucího projektu je dekomponovat projekt na dílčí části - úkoly, ty v programu vytvořit podobně jako se vytváří nový projekt a přidělit je uživatelům k vypracování. K tomu je ovšem potřeba přiřadit k projektu nějaké další uživatele kromě vedoucího. V dané pracovní skupině ve složce All Users klikněte pravým tlačítkem na jméno uživatele, kterého chcete do projektu přiřadit a zvolte možnost Copy user to project. Objeví se okno, ve kterém vyberte cílový projekt a potvrďte volbu kliknutím na Copy.



Obrázek 35 - Přiřazování uživatele k projektu

9. Všichni uživatelé, obsažení ve složce s projektem, nyní mohou být přiřazeni k úkolům, které vytvoří vedoucí projektu.



Obrázek 36 - Uživatelé, kteří jsou přiřazeni k projektu

10. Chcete-li některého z uživatelů z projektu odebrat, klikněte na jeho jméno v daném projektu (viz obrázek 36) pravým tlačítkem a zvolte možnost Remove user from project. Pokud je uživatel řešitelem některého z úkolů v rámci daného projektu, je nutné jej nejprve zbavit této povinnosti.

## B. Konfigurace

### Instalace a nastavení aplikačního serveru

Aplikační server vyžaduje ke správnému fungování pracovní stanici s operačním systémem Windows XP nebo vyšším a platformou .NET verze alespoň 3.5. Dále je nutné disponovat počítačem s instalací SQL Serveru, který podporuje T-SQL. Dle odhadovaného počtu uživatelů aplikace je nutné aplikační server provozovat na stanici s dostatečně velkým pevným diskem, neboť uživatelé mohou na server nahrávat soubory a sdílet je. Jejich maximální velikost lze omezit (popsáno níže). Je-li aplikace provozována v lokální síti, počítač musí mít neměnnou IP adresu. V případě, že je program využíván na internetu, musí mít stanice se serverem veřejnou IP adresu nebo je nutné, aby router, který realizuje překlad adres v rámci lokální sítě, identifikoval pakety také dle portu, na který směřují. To je většinou běžnou vlastností dostupných síťových zařízení.

Aplikační server není nutné instalovat. Všechny soubory, nutné k jeho fungování, se nachází ve složce `install/server` na přiloženém CD. K instalaci tedy stačí celou složku zkopírovat na pevný disk a dodržet následující instrukce. Aplikace sestává ze spustitelného souboru `Server.exe` a knihoven `DataObjects.dll` a `MIME.dll`. Nedílnou součástí je také soubor `Server.exe.config`, který je ve formátu XML a umožňuje měnit základní nastavení:

```
<setting name="BasePort" serializeAs="String">
  <value>14241</value>
</setting>
```

Změnou hodnoty mezi značkami `<value>` a `</value>` dojde ke změně portu, na kterém server realizuje veškeré přenosy dat. Číslo portu se musí pohybovat mezi 1024 a 49151. Dojde-li ke změně tohoto čísla, je nutné změnu provést i u klientských částí aplikace, které se k serveru připojují.

```
<setting name="MaxFileTransferSize" serializeAs="String">
  <value>30000000</value>
</setting>
```

Hodnota 30000000 značí třicet megabytů, což je výchozí omezení velikosti souboru (v bytech), který může uživatel na server nahrát. Číslo je možné libovolně měnit.

```
<connectionStrings>
<add name="Server.Properties.Settings.DEFAULTConnectionString"
connectionString="Data Source=SQLSERVER_NAME; Initial
Catalog=DATABASE_NAME; Persist Security Info=True; User
```

```
ID=login; Password=pass" providerName="System.Data.SqlClient"
/>
</connectionStrings>
```

V uvozovkách za slovem `connectionString` je uveden řetězec, který definuje spojení s databázovým serverem. Řetězec má mnoho jiných přípustných variant a správce databázového serveru by měl být schopný jej vytvořit. Aplikační server se po prvním spuštění pokusí databázi vytvořit (za předpokladu, že neexistuje). Pokud by tento proces selhal, je možné ji vytvořit pomocí přiloženého skriptu s názvem `Database.sql`. Pro provoz na lokálním počítači může mít řetězec například následující tvar:

```
connectionString = "Data Source=PCNAME\SQLEXPRESS;
Initial Catalog = workgroups_database; Integrated
Security=True"
```

Pro správný chod serveru je tedy nutné nastavit komunikační port, který musí být uveden také u klientských částí aplikace. Dále je potřeba správně formulovat řetězec, který obsahuje přístupové údaje do databáze. Pokud se program k databázi nepřipojí, správce o tom informuje. Posledním krokem před spuštěním je umístění programu s knihovnamy do složky, do které je povolen zápis, a které bude zároveň sloužit k uchování souborů, nahrávaných uživateli. Program si soubory udržuje ve vlastní stromové struktuře složek.

## Ovládání aplikačního serveru

Server funguje jako konzolová aplikace. Lze jej ovládat několika textovými příkazy:

- `help`, `h` nebo `?` vypíše na obrazovku veškeré příkazy s vysvětlením, k čemu jsou určeny
- příkaz `new company` si po zadání vyžádá jméno společnosti a vytvoří v databázi o nové společnosti záznam
- `new administrator` vytvoří v rámci společnosti účet správce, který bude spravovat pracovní skupiny. Správců může být ve společnosti víc. Příkaz vyžaduje vložení dalších informací, například názvu společnosti, přihlašovacího jména a hesla správce a další.
- `companies` vypíše seznam společností, které jsou zalistované v databázi.
- `delete company` odstraní z databáze veškeré záznamy spojené se společností, jejíž jméno si příkaz vyžádá.
- `exit` ukončí běh aplikačního serveru. Uživatelé nebudou mít k dispozici aktuální data z databáze a nebude možné stahovat ani nahrávat soubory.

Postup konfigurace aplikačního serveru je jednoduchý. Po spuštění se příkazem `new company` vytvoří nová společnost a v rámci této společnosti příkazem `new administrator` první administrátorský účet, kterým se bude

možné přihlásit prostřednictvím klientské části aplikace a v té začít spravovat uživatelské skupiny. Uvedené příkazy a operace nejsou z klientské části aplikace proveditelné.

## Konfigurace klientské části aplikace

Aplikace potřebuje ke správnému chodu znát IP adresu serveru a port, na kterém server naslouchá. Tyto údaje jsou uloženy podobně jako u aplikačního serveru v souboru `WGAClient.exe.config`, který je ve formátu XML.

Řádky

```
<setting name="BasePort" serializeAs="String">  
    <value>14241</value>  
</setting>
```

definují port, na kterém probíhá komunikace. Hodnota mezi značkami `<value>` a `</value>` musí být stejná jako u aplikačního serveru.

Řádky

```
<setting name="ServerIP" serializeAs="String">  
    <value>127.0.0.1</value>  
</setting>
```

určují IP adresu serveru, ke kterému se klientská část programu připojuje. Tato adresa by měla odpovídat IP adrese stanice, na které je server provozován, případně routeru, který je před něj předřazen. Místo IP adresy je možné zadat doménové jméno počítače.

## C. Obsah CD

- `docs/` – dokumentace k jednotlivým částem programu
- `index.htm` – rozcestník s přehledem obsahu disku
- `install/` – soubory určené k instalaci aplikace
  - `client/` – obsahuje soubory `setup.exe` a `Setup.msi`, určené k instalaci klientské části aplikace
  - `server/` – obsahuje všechny soubory, nutné ke spuštění aplikačního serveru, navíc je přiložen skript `database.sql`, určený k vytvoření databáze
- `source/` – zdrojové kódy programu, včetně všech implementovaných knihoven
  - `client/` – zdrojové kódy klientské části programu
  - `dataobjects/` – zdrojové kódy knihovny, ve které jsou definované základní typy dat, se kterými aplikace pracuje
  - `mime/` – zdrojové knihovny, určené k práci se zprávami ve formátu MIME
  - `server/` – zdrojové kódy aplikačního serveru
- `thesis.pdf` – tento dokument