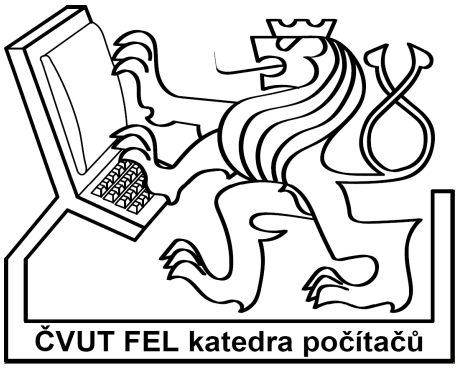
České vysoké učení technické v Praze
Fakulta elektrotechnická




# Bakalářská práce

# Víceuživatelský plánovací kalendář

Vojtěch Outulný

Vedoucí práce: Ing. Pavel Kubalík

Studijní program: Elektrotechnika a informatika, strukturovaný, bakalářský

Obor: Informatika a výpočetní technika

červen 2006

**Poděkování**

Chtěl bych poděkovat vedoucímu bakalářské práce, panu Ing. Kubalíkovi, za jeho vynaložený čas a spolupráci. Dále bych chtěl poděkovat svým rodičům, jenž mě při vzniku této bakalářské práce podpořily.

**Prohlášení**

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 zákona č 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 30.6.2006

.....

**Abstract**

This bachelor work discusses the development of application called Multiuser Scheduling Calendar. The application has three independent parts – client application, server and database.

It is possible to write down meetings, classes, absence and arranged appointments into the calendar. The client application supports more working modes regarding the user (registered vs anonymous). The calendar offers several ways of making a record. The application utilizes its own data format for importing and exporting time schedules with ease. Each user has his/her own calendar, which can be browsed by a validated person. The application enables inserting timetables from HTML files generated by KOS.

The server enables concurrent work of more different users. Communication between the client and the server is encrypted by the Blowfish algorithm. The database implementation is based on MySQL 5.0 server.

**Abstrakt**

Tato bakalářská práce pojednává o vzniku aplikace Víceuživatelský plánovací kalendář. Aplikace je tvořena třemi samostatnými částmi – klientská aplikace, serverová aplikace a databáze.

Do tohoto kalendáře je možné zapsat schůzky, výuku, dobu nepřítomnosti a dohodnuté konzultace. Klientská aplikace podporuje více módů fungování s ohledem na aktuálního uživatele (přihlášený vs anonymní). Tento kalendář umožňuje několik způsobů zápisu. Dále byl navrhnut vlastní formát pro snadné importování a exportování plánů z aplikace. Každý uživatel vlastní kalendář, který může být prohlížen jakoukoliv oprávněnou osobou. Aplikace umožňuje vložení rozvrhu z KOSu prostřednictvím HTML soubor generovaného samotným KOSem.

Server umožňuje souběžnou práci více rozdílných uživatelů. Komunikace mezi klientem a serverem je zabezpečená s využitím šifrovacího algoritmu Blowfish. Databáze je implementována na databázovém serveru MySQL 5.0.

# Obsah

# Seznam obrázků

# 1 Úvod

Cílem této bakalářské práce je vytvoření aplikace Víceuživatelský plánovací kalendář (dále MUSC). V tomto případě se jedná o aplikaci, která bude umožňovat vytvářet časové plány jako běžný papírový plánovací kalendář, ale zde v elektronické podobě. Zároveň je zde kladen velký důraz na snadné a komfortní ovládání, což uživateli umožní rychle a intuitivně vytvářet časové plány dle jeho přání. Aplikace MUSC je tvořena třemi samostatnými komponentami – klientská aplikace, serverová aplikace a databáze.

Klientská aplikace slouží k samotné tvorbě plánů. Tento kalendář umožňuje tvorbu běžných denních událostí jako je schůzka, výuka, doba nepřítomnosti, dohodnuté konzultace a podobně. Dále však aplikace bude poskytovat možnost tvorby tzv. vícedenních událostí, tj. událostí trvajících více než jeden den. Kromě toho bude v programu podpora pro práci s opakujícími se akcemi. Klientská aplikace bude podporovat více módů fungování s ohledem na aktuálního uživatele (přihlášený vs anonymní). Tento kalendář bude umožňovat několik způsobů zápisu. Zapisovat do kalendáře bude možné snadno s pomocí myši a to po předem nastavených časových blocích, nebo pak detailně pomocí přesných časových údajů. Dále bude třeba navrhnout vlastní datový formát pro snadné importování a exportování plánů z aplikace. Tento formát by měl umožňovat snadnou editaci takto exportovaných plánu i bez nutnosti použití aplikace MUSC. Plánovací kalendář bude také umožňovat zobrazení několika překrývajících se akcí a to včetně vyučovacího rozvrhu. Kalendář bude umožňovat vložení rozvrhu z KOSu skrze HTML soubor generovaný samotným KOSem. Plánovací kalendář bude umožňovat pohled na plán pomocí několika různých pohledů, např. Denní plán, Detailní denní plán, KOS plán, Týdenní plán a Měsíční plán.

Každý uživatel bude mít vlastní kalendář, který může být prohlížen jakoukoliv oprávněnou osobou. Samotný plán však může být vytvářen a měněn pouze jeho vlastníkem. Uživatel v situaci, kdy má právo si cizí plán prohlížet, nemá již právo cizí plán jakýmkoliv způsobem měnit. Každý uživatel bude patřit do určitých skupin. Na základě členství v různých skupinách mu budou přidělena příslušná práva na prohlížení cizích plánů.

Server bude umožňovat souběžnou práci více rozdílných uživatelů. Aplikace však nebude podporovat souběžnou práci více uživatelů přihlášených k stejnému účtu. Dále bude třeba zajistit, aby komunikace mezi klientskou a serverovou aplikací byla zabezpečená. Komunikace mezi serverem a databází bude prováděna na základě technologie JDBC. Součástí práce bude vytvoření databáze a její implementace na vhodném databázovém serveru.

Důležitým problémem v současné době je bezpečnost aplikace a snaha o vyvarování se chyb typu „buffer overflow” atd. Další podstatnou otázkou je přenositelnost aplikace mezi různými operačními systémy ať již jsou to systémy typu MS Windows, Linux, Solaris či Mac OS. Tyto otázky byly jedním z hlavních kritérií pro výběr programovacího jazyka v němž bude program implementován a volbě databázového serveru.

# 2 Popis problému, specifikace cílů

## 2.1 Deklarace záměru

Navrhněte Víceuživatelský plánovací kalendář. Tento plánovací kalendář bude umožňovat tvorbu plánů snadnou a intuitivní cestou. Do tohoto kalendáře bude možné zaznamenat schůzky, dobu nepřítomnosti, dohodnuté konzultace atd. Akce bude možné třídit do různých skupin. Každý uživatel bude mít vlastní kalendář a bude členem určitých skupin. Kalendář bude umožňovat současné zobrazení více plánů od různých uživatelů. Pro data bude zvolena jako úložiště MySQL databáze. Celý systém bude nezávislý na použitém operačním systému.

## 2.2 Odborný článek

Cílem této práce je vytvořit Víceuživatelský plánovací kalendář. Tato aplikace má sloužit k tvorbě časových plánů jejích uživatelů. Aplikace bude umožňovat zaznamenat schůzky, školní rozvrh a další akce. Program bude poskytovat několik na sobě nezávislých pohledů na časový plán. Minimálně se očekává Denní plán, Týdenní plán a Měsíční plán. Je možné, aby aplikace nabídla více odlišných pohledů než tyto. Pro každou takovouto aplikaci je bezpodmínečné, aby poskytovala příjemné uživatelské prostředí. Toto prostředí by mělo umožňovat snadnou a rychlou tvorbu plánu. Všechny funkce by měli být snadno a intuitivně dostupné pomocí menu, myši nebo klávesových zkratek.

Bylo by vhodné, aby aplikace poskytovala v nějaké podobě službu importu rozvrhu z KOSu. Jako nejschůdnější variantou se zdá vkládání rozvrhu skrze HTML soubor generovaný samotným KOSem. Pro tuto funkci bude potřeba vytvořit podpůrné nástroje, např. HTML parser.

Každý uživatel bude mít vlastní plán a bude členem určitých skupin. Na základě členství v těchto skupinách získá právo prohlížet plány ostatních uživatelů, kteří náleží do stejných skupinách. Aplikace bude také umožňovat souběžné zobrazení plánů od více uživatelů. Vytvářené akce bude možné třídit do skupin.

Akce lze rozlišovat v plánu podle priority pomocí různých barev. Program bude poskytovat volbu rychlého náhledu na akci. Dále budou jednotlivé pohledy umožňovat zobrazení více současně probíhajících událostí. Akce bude možné tvořit více způsoby. Aplikace bude poskytovat funkci exportu plánu do externího souboru. Hlavním způsobem vytváření akcí bude pomocí myši. Další variantou bude možnost importu plánu z externího souboru. Tento soubor bude také sloužit k přenosu plánů mezi více uživateli. Z tohoto důvodu tedy plyne také požadavek na tento soubor a to, že tento soubor bude sloužit také k exportu plánů z aplikace. Z těchto všech výše zmíněných důvodů plyne, že bude potřeba navrhnout vlastní datový formát. Jedním z hlavních požadavků na aplikaci je nezávislost na použitém operačním systému. Tato skutečnosti jistě bude potřeba vzít v úvahu při návrhu vlastního datového formátu.

Pro aplikaci by byla vhodná architektura klient-server. Serverová aplikace by zajistila komunikaci s klientskou aplikací. Server by dále zajišťoval komunikaci s databázovým serverem, aby se tak zamezilo přímému zásahu uživatele do dat v databázi. Databáze by byla implementovaná na databázovém serveru MySQL. V současné době je velmi podstatnou problematikou ochrana osobních údajů. Z tohoto faktu bude potřeba zajistit bezpečnost komunikace mezi klientem a serverem např. pomocí nějakého šifrovacího algoritmu.

Protože se jedná zcela jistě o složitý a náročný projekt, je proto nutné provést analýzu možného řešení celého problému. Vhodné bude určitě vytvořit seznam uživatelských rolí a jejich popis. Jistě bude také vhodné vytvořit model jednání, aby se určilo jaké mohou být po aplikaci požadované služby jednotlivými uživatelskými rolemi. Dále bude nutné vytvořit ER diagram za účelem vytvoření vhodných struktur v databázi pro uložení dat. V neposlední řadě by byl také vhodný dataflow diagram za účelem určení a sledování datových toků v aplikaci.

## 2.3 Katalog požadavků

Na základě odborného článku byl vytvořen katalog požadavků. Tyto požadavky pokrývají několik samostatných oblastí – databáze, síťová komunikace, návrh GUI, nezávislost na použitém OS, bezpečnost.

1. Navrhnout strukturu dat v databázi.
2. Vytvořit ER diagram dat v databázi.
3. Vytvořit seznam uživatelských rolí a jejich popis.
4. Vytvořit model jednání.
5. Vytvořit dataflow diagram.
6. Zajistit nezávislost aplikace na použitém operačním systému.
7. Implementovat databázi na databázovém serveru Mysl.
8. Vytvořit aplikaci na základě architektury klient-server.
9. Vytvořit serverovou aplikaci, která zajišťuje komunikaci s databází.
10. Vytvořit klientskou aplikaci, která zajišťuje komunikaci se severem a poskytuje GUI pro práci uživatele.
11. Zajistit bezpečnost komunikace mezi klientskou aplikací a serverem.
12. Podpora pro tvorbu plánů v rámci skupiny uživatelů.
13. Možnost seskupit akce do skupin.
14. Navrhnout intuitivní, snadno a rychle použitelné GUI pro klientskou aplikaci.
15. Vytvořit několik pohledů na časový plán (minimálně 3) – např. Denní plán, Týdenní plán a Měsíční plán.
16. Umožnit vložení rozvrhu z KOSu pomocí HTML souboru.
17. Vytvořit HTML parser pro podmnožinu jazyka HTML.
18. Umožnit zobrazení vlastního a cizího plánu současně.
19. Zobrazované akce budou odlišeny v pohledu podle priority barvou.
20. Jednoduchá tvorba a editace akcí pomocí myši.
21. Umožnit více způsobů tvorby akcí např. pomocí myši a prostřednictvím externího souboru.
22. Navrhnout vlastní datový formát souboru pro externí práci s plánem mimo klientskou aplikaci.
23. Poskytnout možnost exportovat rozvrh do externího souboru, který lze použít následně pro přenos plánů mezi více uživateli.

24. Poskytnout možnost importovat plán z externího souboru ve vlastním formátu aplikace.

25. Umožnit rychlý náhled na akci pomocí myši.

26. Schopnost pohledů zobrazit více současně probíhajících akcí.

Nyní se pokusím některé body z katalogu požadavků přesněji specifikovat:

**Ad 6. Zajistit nezávislost aplikace na použitém operačním systému**

V současné době je otázka multiplatformnosti velmi podstatnou problematikou při řešení otázky podnikového softwarového vybavení. Obvyklým jevem je, že např. serverové řešení běží na operačním systému Linux, zatímco osobní stanice zaměstnanců běží na OS Windows. Velký problém pak nastává, když se podnik rozhodne přesunout některou svou část na jinou platformu. Tato situace často vede k nemalým finanční výdajům. Z tohoto důvodu se proto jeví otázka multiplatformity jako podstatná, jelikož může vést k ušetření velkých finančních obnosů. U programu MUSC se jeví jako velmi pravděpodobné, že serverová aplikace poběží pod operačním systémem Linux. Naopak klientská aplikace poběží buď na platformě Windows, Linux nebo Mac OS. Bylo by zcela jistě nerozumné vyvíjet pro každou platformu aplikaci samostatně. Požadavek multiplatformnosti byl proto zcela logicky vztažen i na ostatní části aplikace.

**Ad 8. Vytvořit aplikaci na základě architektury klient-server**

Tento požadavek byl vznesen z důvodu bezpečnosti. Byla zde nastíněna otázka zda je vhodné, aby uživatel mohl přímo manipulovat s údaji v databázi. Situace, kdy by uživatel mohl přímo zasahovat do databáze by mohla vést k bezpečnostním problémům.

**Ad 12. Podpora pro tvorbu plánů v rámci skupiny uživatelů**

Program by měl umožňovat třídit akce do skupin. Dále celý systém umožňuje, aby jednotliví uživatelé mohli náležet do různých skupin. Na základě členství v těchto skupinách by uživatelé získali právo náhledu na akce ostatních uživatelů, kteří náleží do stejné skupiny. Tato funkce by pak umožňovala vytvářet a sdílet plán v rámci určité skupiny uživatelů. Díky této funkci by uživatelé mohli být zároveň členy více skupin.

**Ad 14. Navrhnout intuitivní, snadno a rychle použitelné GUI pro klientskou aplikaci**

Určitě žádný uživatel nemá rád, když musí v aplikaci nějakou funkci dlouho hledat nebo je nucen číst si manuál. Z tohoto důvodu je potřeba navrhnou GUI, které poskytne intuitivní prostředí, ve kterém uživatel vždy ví, kde má co hledat a samozřejmě to tam nalezne. Důležitá je také přehlednost prostředí, kdy uživateli musí být hned zřejmé k čemu slouží jednotlivé ovládací prvky a funkce. Dále je potřeba zajistit, aby jednotlivé funkce byly logicky roztříděny do skupin a uživatel rychle věděl, kde začít. Zároveň je nutné, aby jednotlivé dialogy byly přehledně uspořádány a vždy nabízely pouze optimální množství funkcí. Jelikož příliš široké spektrum funkcí nebo naopak příliš úzké, by mohlo uživatele odradit od užívání aplikace.

**Ad 15. Vytvořit několik pohledů na plán (minimálně 3) – např. Denní plán, Týdenní plán a Měsíční plán**

Aby program sloužil svému účelu, je třeba aby poskytoval několik různých pohledů na plán. Tyto pohledy musí vždy poskytnout jasný pohled na dané časové období. V rámci tohoto období je potřeba zajistit dostatečnou přehlednost o událostech probíhající v nejbližších termínech. Protože uživatel vytváří akce, které trvají krátký časový okamžik, je vhodné aby byl přítomen i Detailní denní plán. Velmi často také při domluvě o schůzce bereme v potaz akce, které se konají v nejbližší dny. Z tohoto důvodu by byl jistě vhodný Týdenní plán. Není však žádnou zvláštností, že

občas plánujeme, akce které trvají více dní, např. dovolená nebo pracovní cesta. V této situaci je zcela potřeba vzít v potaz události odehrávající se ve větším časovém období. V tomto případě by byl zcela na místě Měsíční plán.

**Ad 16. Umožnit vložení rozvrhu z KOSu pomocí HTML souboru**

Každodenní součástí každého zaměstnance ČVUT nebo jejího studenta je výuka. Tato skutečnost musí být jistě vzata v potaz při plánování celé řady akcí.

**Ad 18. Umožnit zobrazení vlastního a cizího plánu současně**

Jak již bylo dříve zmíněno, v programu by měla být zabudována podpora pro tvorbu plánů v rámci skupin. Proto je celkem logickým požadavkem, aby uživatel mohl svůj plán korigovat s plánem v rámci určité skupiny. Z tohoto důvodu je však potřeba mít zobrazeno současně více rozvrhů od různých lidí. V zájmu bezpečnosti by bylo vhodné, aby se cizí akce daly prohlížet, ale ne už editovat či dokonce mazat. Díky tomu, že se vždy zobrazí cizí události náležící do určitých skupin, lze povolit i tzv. anonymní uživatele. Zároveň tato funkce umožní anonymním uživatelům prohlížet si např. pouze školní rozvrh registrovaných uživatelů a veřejné schůzky .

**Ad 21. Umožnit více způsobů tvorby akcí např. pomocí myši a prostřednictvím externího souboru**

Jednotlivé události v kalendáři je potřeba nějakým způsobem vytvořit. Každému uživateli vyhovuje jiný styl tvorby plánů. Dále je také třeba vzít v potaz, že některé styly tvorby událostí se nehodí pro všechny případy, např. jednorázová tvorba velkého množství akcí. Proto je potřeba uživateli nabídnout více způsobů tvorby akcí.

Asi nejpřirozenějším způsobem vytvoření akce je tvorba pomocí myši. V tomto případě uživatel klikne myší a pomocí tažení myší označí oblast symbolizující období na něž má být akce vytvořena. Následně se zobrazí jednoduchý a přehledný dialog a umožní vznik nové akce. Tento způsob tvorby akce je vhodný pro tvorbu jednotlivých akcí s možností jejich okamžitého náhledu. Naopak tento způsob naprosto nevyhovuje pro jednorázové vytvoření velkého množství akcí.

V tomto případě by jistě bylo vhodné poskytnout standardizované a jednoduché rozhraní pro tvorbu velkého množství akcí. Vhodným řešením by byla možnost importu akcí z externího souboru. Tento soubor by musel mít snadný a přehledný formát pro tvorbu akcí. Zároveň je třeba vzít v potaz skutečnost, že tento soubor a jeho formát musí být opět platformně nezávislý.

# 3 Rešerše stávajících řešení

Pro rešerši jsem vybral dva komerční projekty – Microsoft Outlook 2000 a Calendar Quick v3.1. MS Outlook 2000 jsem vybral, protože je to představitel produktu od velké a renomované firmy, za jehož vývojem stojí velký tým a nemalé finanční prostředky vynaložené na jeho vývoj. Proto lze předpokládat vysokou profesionální úroveň a širokou paletu nabízených funkcí. Důvodem pro nepoužití dnes již existující verze MS Outlook 2003 byla skutečnost, že se mi nepodařilo tento produkt získat ani jsem k němu neměl přístup. Na druhé straně stojí program Calendar Quick, jehož autorem je jediná osoba Brian K. Holdsworth.

## 3.1 Rešerše Microsoft Outlook 2000

MS Outlook 2000 je program, který neslouží pouze jako plánovací kalendář. Tento program poskytuje celou škálu služeb. Na levé straně programu je svislá lišta, která poskytuje jednoduché navigační menu. Zde může uživatel nalézt nejčastěji používané služby, jako je Outlook dnes, kalendář, kontakty, úkoly, poznámky a odstraněná pošta. Protože služba „odstraněná pošta" nesouvisí s tématem bakalářské práce, nebudu se jí již dále zabývat. Veškeré údaje jsou v Outlooku šifrovány za využití 40 bitového klíče.

Služba Outlook dnes poskytuje rychlý přehled akcí zaznamenaných do kalendáře a úkolů pro aktuální den. Je zde vždy seznam všech akcí a úkolů na něž lze najet kurzorem a po kliknutí na ně se otevře okno pro editaci příslušné akce či úkolu.

Na záložce kalendář lze nalézt tři samostatné části – tabulku na zaznamenávání úkolů, panel úkolů a měsíční kalendář. Tabulka na plánování akcí zobrazuje akce minimálně s 30 minutovými intervaly trvání. Po kliknutí do tabulky se zobrazí okno pro editaci akce. Do kalendáře lze umístit více souběžně probíhajících akcí. Tyto akce se pak zobrazují vedle sebe a jejich velikost se přizpůsobuje vzhledem k pevně určené šířce tabulky, což při více současně probíhajících událostech může působit nepřehledně. V dialogu pro editaci akce lze určit několik položek u každé akce. Pro každou akci lze určit název (text následně zobrazený v plánovacím kalendáři), umístění, počáteční datum a čas akce, konečné datum a čas akce a nakonec samotný text akce. Datum u akcí lze určit dvěma způsoby, buď vepsat datum ručně nebo s využitím kalendáře. Čas u akcí lze volit pouze po předem daných půlhodinových blocích. Situace, kdy dojde k zadání chybných dat u data akce, někdy skončí upozorněním na chybu, ale někdy program tuto chybu ignoruje. Např. správně „Sun 3/19/2006" a špatně „Sun 3/19/=2006". I přes tuto chybu je akce vytvořena. U každé akce si lze určit zda se má akce opakovat. Lze si vybrat zda se má akce opakovat denně, týdně, měsíčně nebo ročně. Pro každé opakování lze určit rozsah opakování. Tento rozsah lze určit jako neomezený, konečným datem anebo počtem opakování. Pro každou akci lze určit zda na ní chcete upozornit. Outlook umožňuje čtyři základní pohledy na plán – denní, pracovní týden (Po – Pá), týdenní (Po – Ne) a měsíční. Aplikace umožňuje akce třídit do skupin a vyhledávání akcí podle názvu nebo textu v nich obsažených.

V měsíčním kalendáři jsou zobrazeny dva po sobě jdoucí měsíce. Nelze zobrazit dva po sobě nejdoucí měsíce. Vybráním dne v kalendáři se změní pohled v tabulce na akce pro příslušný den. Poslední samostatnou částí je panel úkolů, který zobrazuje úkoly pro aktuální den. Tyto úkoly lze editovat po kliknutí na ně.

Na záložce kontakty lze snadno vytvářet kontakty na osoby. Po kliknutí se zobrazí dialog pro vytvoření nového kontaktu. U každého kontaktu lze vyplnit všechny potřebné informace pro danou osobu. Kontakty lze třídit podle abecedy do jednotlivých sekcí.

Další službou je tvorba úkolů. Záložka je tvořena jednou tabulkou obsahující seznam úkolů. Úkoly lze snadno vytvořit pomocí kliknutí do tabulky. Samotné úkoly jsou v podstatě velmi podobné akcím v kalendáři. Úkoly jsou vlastně akce bez bližšího časového určení (hodina a minuta). U každého úkolu je třeba zadat datum zahájení a datum splnění. Ostatní položky jsou stejné jako u akce v plánovacím kalendáři. Poslední sekcí je možnost tvorby poznámek. Jedná se o možnost zaznamenání textu (poznámky) bez jakýchkoliv bližších časových údajů.

Rešerše byla prováděna na české verzi programu Microsoft Outlook 2000, který běžel

na anglické verzi operačního systému MS Windows XP Pro SP2 s nainstalovanou podporou pro český jazyk. Bohužel se vyskytly problémy s českou lokalizací programu. Diakritika u některých českých slov na ovládacích prvcích v Outlooku byla zkomolená, a proto některé výrazy byly nečitelné a znesnadňovaly tak ovládání.

## 3.2 Rešerše Calendar Quick v3.1

Calendar Quick je na rozdíl od programu Microsoft Outlook 2000 v zásadě pouze plánovací kalendář. Tento program umožňuje několik pohledů na plán – měsíční, týdenní, denní, časovou akci, plánovací a seznam akcí. Dále program umožňuje vyhledávat v již existujících akcích. Jednou z dalších možných funkcí je filtrování výpisu a zobrazování akcí na základě členění akcí do pěti skupin. Akce musí patřit pouze do jedné z následujících skupin – „events", „phases", „deadlines", „anniversaries" a „actions".

U všech pohledů lze snadno vytvářet nové akce. Stačí vybrat požadovaný časový okamžik a kliknout pravým tlačítkem myši. Následně je uživatel povinen zařadit akci do jedné z pěti skupin. Po vybrání skupiny se zobrazí dialog pro editaci nové akce. U akce (action) lze určit její název, počáteční a koncové datum, text a prioritu. Datum lze určit pouze pomocí kalendáře, nelze ho vepsat ručně. U akce nelze určit přesné časové určení (hodina a minuta), je určeno již výběrem časového období.

Pokud si uživatel zvolí akci ze skupiny „event", může u ní určit přesný počáteční a koncový čas. Nevýhodou je však, že „event" může být pouze v rámci jednoho dne, a tak nelze vytvořit akci začínající první den v přesný okamžik a končící druhý den ve stejný okamžik. Dále lze u „event" určit název, text a možnost případného upozornění. U „event" lze určit zda se má akce opakovat. Lze zvolit buď opakování po jednotlivých dnech v týdnu nebo opakování s pevným intervalem anebo opakování po měsíci s počátečním dnem. Podobně jako Outlook má i tato aplikace problémy s kontrolou vstupů. Např. při zadání počátečních a koncových časů „rrrrrrr", vytvoří bez upozornění uživatele akci trvající nulový časový okamžik např. „10:00a – 10:00a". Nemožnost vytvořit vícedenní událost s přesným určením počátečního a koncového času neumožňuje ani akce ve skupině „phases" s jejíž pomocí lze vytvářet akce trvající více dní. Akce lze zobrazovat minimálně po pevných pětiminutových blocích. U každého plánu lze přejít na další logický blok (zobrazený časový úsek vzhledem k danému pohledu) pomocí tlačítek „Prev" a „Next".

## 3.3 Shrnutí rešerše

Obě výše popisované aplikace Microsoft Outlook 2000 a Calendar Quick v3.1 poskytují několik pohledů na plán. Některé aplikace mají více možných pohledů, ale běžnými jsou pohledy – denní, týdenní a měsíční. Každá aplikace se snaží umožnit třídit data do skupin. Outlook seskupuje akce do vlastních skupin a toto zařazení lze kdykoliv změnit. Naopak Calendar Quick třídí akce hned při jejich vytvoření do jedné z pěti skupin, kterou pak již nelze měnit. U Outlooku lze vytvářet nové vlastní skupiny, ale u Calendar Quick tato možnost není. Obě aplikace poskytují plánování po pevných blocích, ať již u Outlooku po 30 minutových nebo u Calendar Quick po 5 minutových blocích. Oba zástupci umožňují opakování akcí dle několika hledisek a upozornění na akci. Zajímavou skutečností je, že obě aplikace mají problémy s kontrolou vstupních dat. Tyto chyby naštěstí nezpůsobí pád aplikace. U MS Outlook je tento problém menší, naopak u aplikace Calendar Quick je správná činnost závislá v podstatě na uživateli.

U aplikace Microsoft Outlook 2000 bych vyzvedl určitě velkou nabídku různého nastavení u akcí. Naopak u aplikace Calendar Quick v3.1 lze pochválit velmi dobrou nápovědu při samotné práci, kdy i úplný začátečník je schopen rychle vytvořit plán dle svých požadavků. Aplikace Outlook je určena spíše pro zkušené uživatele, kteří si chtějí u akcí nastavit hodně parametrů. Dále lze u Outlooku kladně hodnotit pestrou paletu způsobů plánování, např. akce v kalendáři, úkoly či poznámky. Naproti tomu Calendar Quick je vhodný pro začínající uživatele. Hlavním důvodem je především větší „přímočarost" v tvorbě plánu a všudypřítomná nápověda. Nevýhodou pro některé uživatele může být fakt, že program je dostupný pouze v anglickém jazyce.

# 4 Analýza řešení

## 4.1 Uživatelské role

S aplikací bude pracovat mnoho lidí. I přes tuto skutečnost lze vysledovat určité skupiny lidí, kteří provádějí jisté činnosti opakovaně. Na základě těchto skupin lze vytvořit tři uživatelské role s určitými přidělenými právy pro práci se systémem.

### 4.1.1 Anonymní uživatel (AU)

Anonymní uživatel (dále AU), nemá žádná práva nad databází. Pokud AU zná uživatelské jméno některého registrovaného uživatele, může si prohlížet jeho akce, které náleží do skupiny ALL a KOS. AU nemůže do plánů přidávat akce nebo je z nich odebírat ani je nemůže nijak měnit. AU nemá právo rozvrh uložit do databáze. AU nemůže systém požádat o import vlastního nebo cizího plánu z lokálního disku ani o import rozvrhu z KOSu. AU má právo exportovat plán ze systému, avšak pouze akce náležící do skupiny ALL a KOS. AU si může zobrazit podrobnosti o vybrané akci. AU má možnost zobrazení kalendáře. AU se může přihlásit do systému nebo může požádat o vytvoření nového uživatelského účtu. AU nemá právo prohlížet si cizí osobní údaje.

### 4.1.2 Přihlášený uživatel (PU)

Přihlášený uživatel (dále PU) implicitně náleží do skupiny PRIVATE, ALL a KOS. Do dalších skupin může být PU přidán pouze administrátorem. Po přihlášení do systému, může PU systém požádat o zobrazení vlastního rozvrhu. PU může také požadovat zobrazení cizího rozvrhu pokud zná příslušné uživatelské jméno. Přihlášenému uživateli jsou zobrazeny pouze události cizího uživatele, které náleží do okruhu skupin, který vznikne průnikem skupin, do nichž oba uživatelé náleží (kromě skupiny PRIVATE). PU může skrze systém provádět pouze změny akcí, které náleží jemu, cizí události PU nemůže měnit ani mazat. PU může vytvářet nové akce, které náleží pouze jemu a určit do které skupiny náleží. PU má právo importovat plán, který je ve vlastním formátu aplikace a případně ho měnit, smazat nebo uložit do databáze. PU má právo importovat rozvrh z KOSu, který je v HTML souboru generovaný samotným KOSem. Tento rozvrh může PU případně měnit, smazat nebo uložit do databáze. PU smí exportovat jakýkoliv plán, který je schopen s ohledem na práva, získaná členstvím ve skupinách, zobrazit, na lokální disk klienta. PU nesmí měnit soubor s inicializačními daty na straně serveru. PU si může zobrazit podrobnosti o vybrané akci. PU má možnost zobrazit si kalendář. Přihlášený uživatel se může přihlásit do systému nebo může požádat o vytvoření dalšího uživatelského účtu. PU má právo prohlížet si své osobní údaje a členství ve skupinách. Dále má právo uložit si své přihlašovací údaje pro možnost „rychlého přihlášení“.

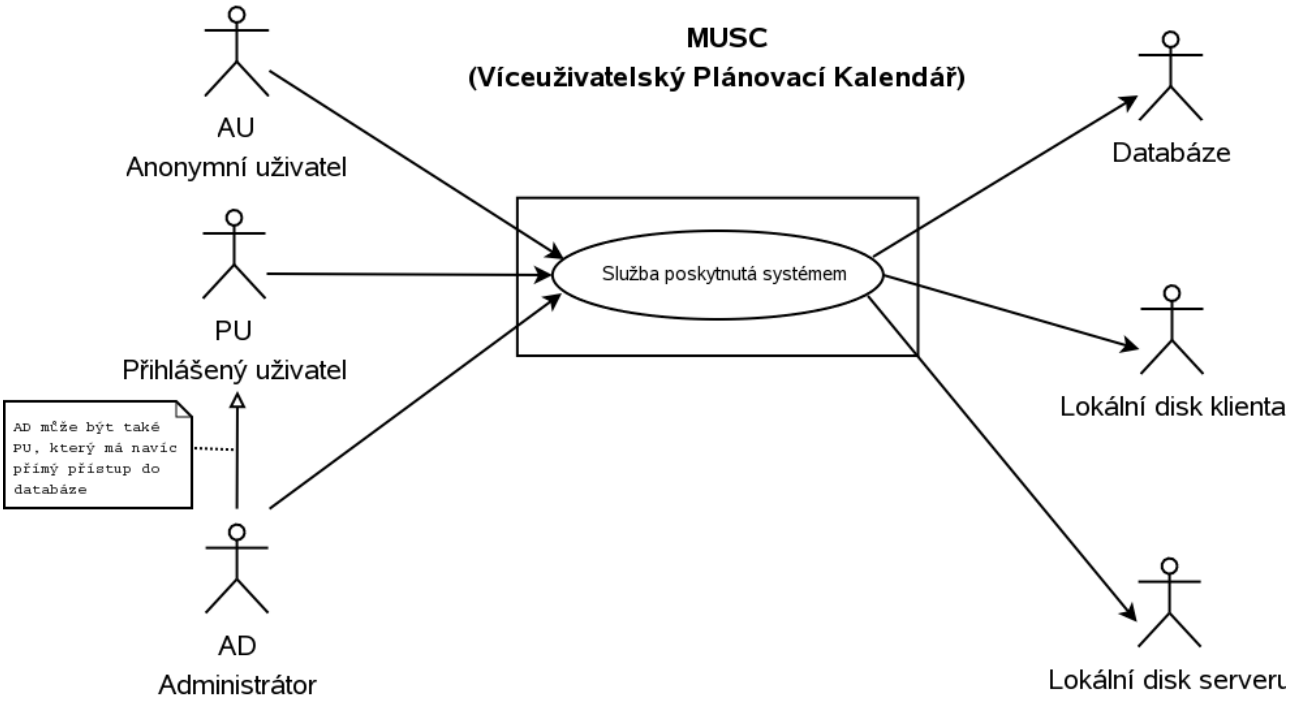
### 4.1.3 Administrátor (AD)

Po přihlášení do systému má administrátor (dále AD) stejná práva jako PU. AD má navíc možnost provádět změny dat v databázi prostřednictvím přímého přístupu do databáze, neděje se tak skrze systém. AD má právo přidávat PU do dalších skupin. AD je správcem systému, jak aplikace na straně serveru, tak i databázového serveru. AD nezodpovídá za klientskou část aplikace. AD smí měnit soubor s inicializačními daty na straně serveru, která obsahují např. informace potřebné k připojení do databáze.

## 4.2 Model jednání

UML diagram modelu jednání naleznete v příloze pod číslem 1. V modelu jednání se vyskytuje samotný systém MUSC a pět aktérů – přihlášený uživatel (PU), anonymní uživatel (AU), administrátor (AD), databáze, lokální disk klienta a lokální disk serveru. Zjednodušený model jednání je vidět na obrázku 1.

AU má pouze omezená práva. AU může systém požádat o novou registraci (vytvoření nového uživatelského účtu) nebo se přihlásit, pokud má registraci již z dřívější doby. Při registraci musí uživatel zadat všechny povinné údaje – vlastní jméno a příjmení, uživatelské jméno, heslo, školní e-mail a typ uživatele. Typ uživatele může být např. student, učitel, zaměstnanec školy apod. Uživatel může dále vyplnit nepovinné údaje jako je soukromý e-mail nebo telefon. Systém následně předá registrační údaje databázi a poté je uživatel zaregistrován. V tento okamžik se neregistrovaný uživatel stává registrovaným uživatelem a přechází ze stavu AU do stavu PU. AU má právo požádat systém o zobrazení cizího plánu. Systém předá jeho požadavek databázi a pokud příslušné uživatelské jméno osoby, jejíž plán má být zobrazen je správné, je mu plán zobrazen. Jsou mu však zobrazeny pouze akce, které náleží do skupiny ALL a KOS. AU může také systém požádat o export zobrazených akcí na lokální disk klienta. AU nemá právo vytvářet akce, editovat je či mazat.




**Obrázek 1: Zjednodušený model jednání**

PU může systém požádat o vytvoření dalšího uživatelského účtu nebo se přímo přihlásit. Pokud PU požádá o novou registraci, je mu vytvořen další účet. PU může systém požádat o zobrazení vlastního nebo cizího plánu. Následně systém požádá databázi o poskytnutí potřebných dat pro zobrazení plánů. Poté co systém potřebná data obdrží, vykreslí požadovaný plán. PU jsou však poskytnuty jen ty akce, které náleží do průniku skupin, jejichž členem je PU a cizí uživatel, jehož plán chtěl PU zobrazit (vyjma akcí náležících do skupiny PRIVATE). Když má PU zobrazen vlastní plán, může požádat systém o přidání události, odebrání události nebo změna události. Následně po potvrzení změn v plánu uživatelem, jsou změněná data uložena do lokální paměti. Až při provedení volby „Uložit plán" jsou data uložena do databáze.

Jednou ze služeb jež systém poskytuje je import plánu nebo import rozvrhu generovaného KOSem. PU požádá systém o import plánu a oznámí mu umístění plánu na lokálním disku. Plán musí být ve vlastním formátu aplikace. Po zadání potřebných údajů je plán načten z lokálního disku klienta. Následně je importovaný plán zobrazen aplikací. Pokud chce PU importovat rozvrh z KOSu, musí požádat systém o import rozvrhu z KOSu. Následně mu oznámí umístění souboru s rozvrhem na lokálním disku klienta. Rozvrh z KOSu musí být v HTML formátu, který je generována samotným KOSem. Následně je importovaný rozvrh zobrazen aplikací.

Další službou, o kterou může PU požádat, je export zobrazeného plánu. Exportován může být buď plán, který náleží pouze PU nebo jen cizí plán anebo obě dvě varianty zároveň. Po přijmutí požadavku systémem, je plán uložen na lokální disk klienta ve vlastním formátu aplikace. PU má možnost požádat systém o uložení svých identifikačních (přihlašovacích) údajů na lokální disk. Tato volba je vhodná pro tzv. „rychlé přihlášení".

AD má stejné pravomoce jako PU. AD má navíc možnost přímého zásahu do databáze, který se však neděje skrze aplikaci MUSC.

Při spuštění aplikace na straně klienta, jsou nejprve systémem načtena inicializační data z inicializačního souboru na lokálním disku klienta, která obsahují informace o nastavení aplikace. Poté je navázáno spojení se serverovou částí aplikace.

Při spuštění aplikace na straně serveru, jsou nejprve systémem načtena inicializační data z inicializačního souboru na lokálním disku serveru, která obsahují např. informace potřebné k připojení k databázi. Poté je navázáno spojení s databází.

## 4.3 ER model dat v databázi

ER diagram dat v databázi naleznete v příloze pod číslem 2. Zároveň v příloze pod číslem 3 naleznete legendu k ER diagramu. V databázi jsou čtyři základní entity – UZIVATEL, AKCE, SKUPINA a UZIVATEL_SKUPINA. Registrovaný uživatel je v databázi reprezentován entitou UZIVATEL. O uživateli je potřeba shromáždit několik povinných údajů, které přispějí k jednoznačné identifikaci samotného uživatele a zároveň i k bezpečnosti celého systému. Povinné údaje jsou jméno a příjmení, heslo a školní e-mail. Aby byla zaručena jednoznačná identifikace uživatele je potřeba zadat unikátní uživatelské jméno, které je uloženo v atributu UZIVATELSKE_JMENO. Každý uživatel má přiděleno unikátní identifikační číslo ID_UZIVATEL. Každá AKCE je identifikována unikátní číslem ID_AKCE a číslem skupiny opakujících se akcích ID_OPAKOVANI v rámci uživatele, který ji vytvořil. Proto je u každého uživatele povinný atribut DALSI_ID_AKCE, který obsahuje identifikační číslo pro příští vytvořenou akci, a atribut DALSI_ID_OPAKOVANI, který určuje příští identifikační číslo pro skupinu opakujících se akcí. Uživatel může do databáze zadat další, již nepovinné, údaje, jako je telefon nebo soukromý e-mail, které lze využít např. pro jeho snadné kontaktování v případě problémů.

Entita AKCE reprezentuje jakoukoliv událost zaznamenanou v plánu. Každé události je přiděleno unikátní číslo ID_AKCE, aby mohla být jednoznačně identifikována v rámci uživatele, který ji vytvořil. U každé události je potřeba zjistit, který uživatel ji vytvořil, a proto obsahuje atribut ID_UZIVATEL. Každá akce je pak jednoznačně identifikovatelná v rámci celé databáze pomocí unikátní dvojice atributů ID_AKCE a ID_UZIVATEL. Tato vlastnost je zaručena vztahem UZIVATEL_UDÁLOST, kdy každá událost mohla být vytvořena pouze jedním uživatelem, zatímco uživatel může vytvořit žádnou nebo více událostí. U každé události je potřeba sledovat prioritu (např. vyšší a nižší priorita), určena atributem PRIORITA, a typ akce (např. konzultace nebo vyučování), určená atributem TYP_AKCE. Uživatel může vytvořit událost, která se pravidelně opakuje. Tato opakující se událost je následně rozkopírována do databáze po časové období, kdy se událost opakuje. Aby bylo možné např. všechny opakující se události odstranit naráz, je třeba pro každou takto vytvořenou skupinu opakujících se událostí určit číslo skupiny ID_OPAKOVANI. Událost musí samozřejmě obsahovat samotný poznamenaný text k události, který je uložen v atributu TEXT. U události dále sledujeme počáteční rok, měsíc, den, hodinu a minutu a koncový rok, měsíc, den, hodinu a minutu, což odpovídá atributům START_ROK, START_MESIC, START_DEN, START_HODINA, START_MINUTA a KONEC_ROK, KONEC_MESIC, KONEC_DEN, KONEC_HODINA a KONEC_MINUTA.

Entita SKUPINA obsahuje informace o všech událostech a specifikuje u nich do jaké náleží skupiny. Skupina je seskupení více uživatelů pod jeden sjednocující prvek (skupinu). Každá akce může náležet pouze do jedné skupiny. Každý záznam je jednoznačně identifikován atributem ID_SKUPINA. Skupina do níž daná akce náleží je určena atributem SKUPINA.

Entita UZIVATEL_SKUPINA obsahuje informace do jakých skupin daný uživatel náleží. Tato entita má dvě položky – ID_UZIVATEL (primární klíč entity UZIVATEL) a SKUPINA. Unikátní klíč entity UZIVATEL_SKUPINA je tvořen dvojicí atributů ID_UZIVATEL a SKUPINA. Po zaregistrování získá každý uživatel členství ve skupině PRIVATE, ALL a KOS. Do dalších skupin může být uživatel přidán pouze administrátorem. Do skupiny PRIVATE náleží všechny soukromé akce PU, které nemůže zobrazit nikdo cizí. Do skupiny ALL patří všechny běžné akce, které mohou být zobrazeny ostatními PU. Do skupiny KOS náleží školní rozvrh, tyto akce jsou viditelné pro ostatní PU. Anonymní uživatel si může nechat zobrazit pouze akce, které náleží do skupin ALL a KOS

## 4.4 Dataflow diagram

Systém má čtyři základní zdroje dat – lokální disk klienta, vyrovnávací paměť na straně klienta, lokální disk serveru a databázi. Na obrázku 2 lze vidět dataflow diagram pro celou aplikaci. Dále se systémem spolupracují tři samostatní aktéři – anonymní uživatel (AU), přihlášený uživatel (PU) a admonistrátor (AD). Při spuštění aplikace na straně klienta je potřeba načíst inicializační data, která obsahují nastavení klientské aplikace. Tato inicializační data jsou uložena na lokálním disku klienta v konfiguračním souboru. Pro zprovoznění aplikace na straně serveru je také potřeba získat inicializační data, která obsahují např. informace potřebná pro připojení do databáze. Tato inicializační data jsou uložena v souboru s inicializačními údaji na lokálním disku serveru.

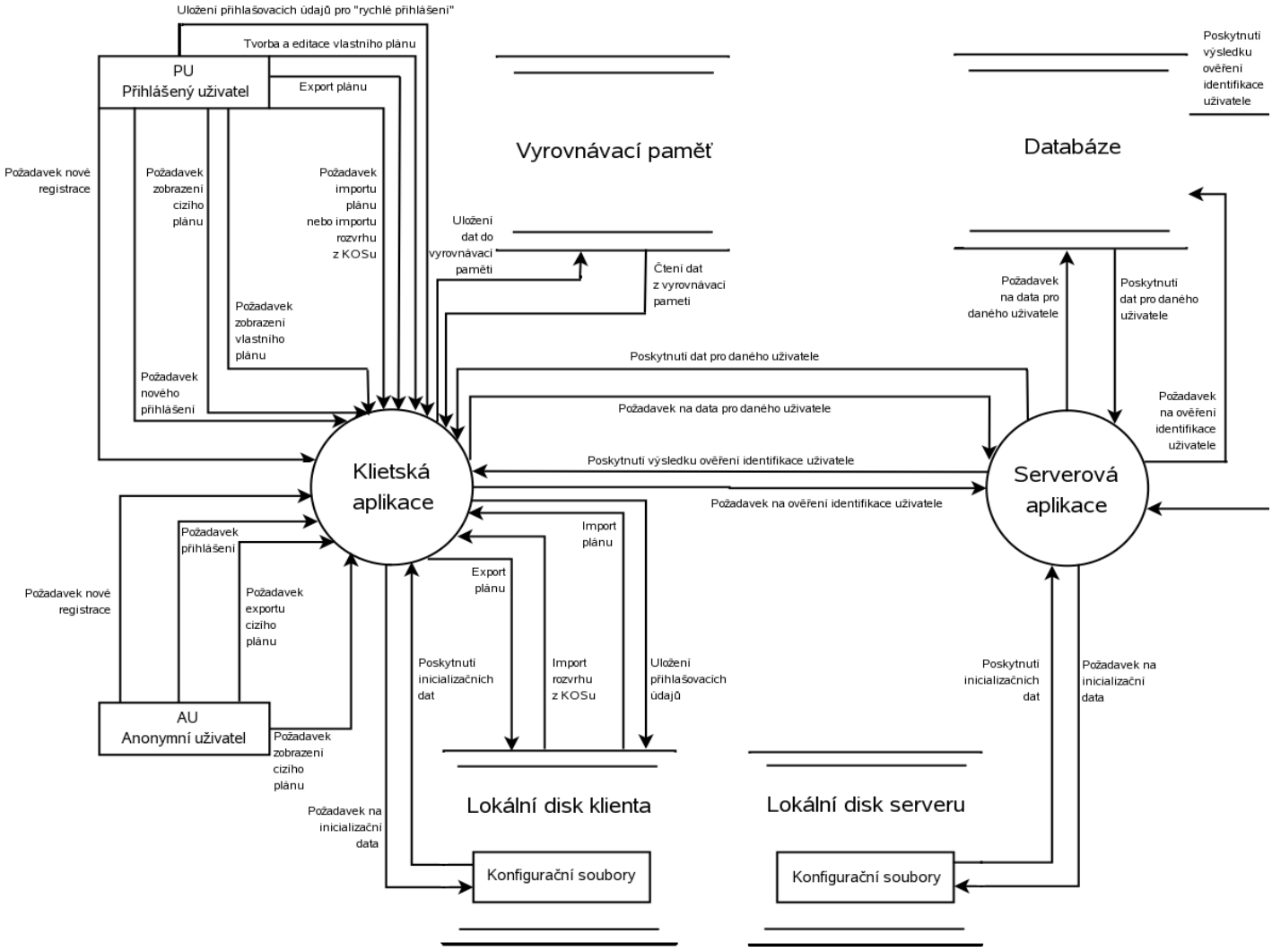
AU může klientskou část aplikace (dále systém) požádat o novou registraci a následně poskytnout potřebná data pro registraci. V tomto případě předá systém požadavek na registraci společně s poskytnutými daty serveru. Server předá data databázi. Databáze vrátí serveru potřebná data a na základě těchto dat server rozhodne o požadavku. Výsledek rozhodnutí o požadavku společně s případnými daty z databáze vrátí server klientské aplikaci. AU může systém také požádat o zobrazení cizího rozvrhu. AU jsou zobrazeny pouze akce jenž náleží do skupiny ALL a KOS. Tato žádost je zpracována obdobným způsobem jako žádost o registraci. Pokud se chce AU přihlásit, aby se tak mohl stát PU, musí systém o tento úkol požádat. Systém předá serveru požadavek na přihlášení. Server příjme přihlašovací údaje a porovná je s údaji uloženými v databázi. Bez ohledu na to, zda identifikační údaje byly správné či ne, server pošle systému informaci o výsledku operace přihlášení. AU může systém požádat o export zobrazeného plánu. Systém následně uloží plán ve vlastním formátu aplikace na lokální disk klienta.

PU má stejné možnosti jako AU, a tak může systém požádat o vytvoření dalšího uživatelského účtu, opětovné přihlášení nebo export zobrazeného rozvrhu. Tyto události probíhají stejně jako u AU. Zvláštním případem je možnost požádat o zobrazení cizího rozvrhu. Pokud je uživatel pouze AU jsou mu poskytnuty pouze akce ze skupin ALL a KOS. Naproti tomu PU má možnost zobrazení cizích událostí z více skupin. Pokud PU požádá systém o zobrazení cizího plánu, jsou PU poskytnuty všechny akce, které náleží do průniku skupin do nichž patří PU a cizí uživatel, jehož události mají být zobrazeny (do průniku skupin nenáleží skupina PRIVATE). Tento požadavek je předán klientskou částí serveru. Server vykoná příslušný dotaz do databáze a získaná data vrátí systému. PU má právo požádat systém o vytvoření nových akcí, které by náleželi PU. PU smí také systém požádat o editaci a případné smazání akcí, které patří PU. Všechny změny plánu jsou ukládány do vyrovnávací paměti. Tyto změny jsou uloženy teprve až PU požádá systém o uložení těchto změn. Systém pošle požadavek na uložení plánu a data k uložení na server. Server poté provede uložení dat do databáze. Pokud PU neprovede uložení těchto změn a program ukončí, budou všechny změny nenávratně ztraceny. PU může chtít využít volby tzv. „rychlého přihlášení“. Aby mohl uživatel při přihlášení využít tuto možnost je třeba nejprve požádat systém o uložení identifikačních údajů. Systém uloží identifikační údaje v zašifrované podobě do souboru na lokální disk klienta.

Další možností PU je import vlastního nebo cizího plánu. PU požádá systém o tuto volbu a

sdělí mu umístění souboru s daty na lokálním disku klienta pro importovaní. Data pro import musí být ve vlastním formátu aplikace. Systém poté provede načtení dat ze souboru a uloží je do vyrovnávací paměti. Tato data jsou natrvalo uložena až po volbě „uložit plán“. PU smí také požádat systém o import rozvrhu z KOSu. Rozvrh musí být v HTML souboru a ve formě generované přímo KOSem. PU musí sdělit systému informaci o umístění souboru s rozvrhem z KOSu na lokálním disku klienta a časové období na které má být rozvrh importován. Systém poté provede kontrolu souboru a oznámí, zda je soubor v pořádku. Pokud je soubor v pořádku je proveden import rozvrhu. Systém poté provede načtení dat ze souboru a uloží je do vyrovnávací paměti. Tato data jsou natrvalo uložena až po provedení volby „uložit plán“.

AD má po přihlášení stejná práva jako PU. AD má navíc možnost přímého přístup do databáze, což se neděje skrze aplikaci MUSC.




**Obrázek 2: UML diagram – dataflow**

# 5 Návrh implementace

## 5.1 Implementační prostředí

Důležitou otázkou pro návrh programu je i volba vhodných implementačních nástrojů. V našem případě se jedná o programovací jazyk a databázový server.

### 5.1.1 Volba programovacího jazyka pro implementaci

Otázka výběru vhodného programovacího jazyka, který bude následně použit pro implementaci, je velmi důležitá a nesmí být podceněna. Nakonec jsem se rozhodl pro programovací jazyk Java. Tento jazyk má sice řadu nevýhod, ale také mnoho pozitivních stránek.

Mezi negativní vlastnosti jistě patří fakt, že se jedná o interpretovaný programovací jazyk. Obecně se soudí, že interpretovaný jazyk musí být nutně pomalý. Tento fakt však v současnosti již ne zcela odpovídá realitě. Problém u interpretovaných jazyků spočívá v práci s pamětí. Tuto činnost lze rozdělit na dvě samostatné činnosti: alokace paměti a uvolňování paměti. Alokace paměti je u interpretovaných jazyků s využitím garbage collectoru pozitivní vlastností, jelikož jsou v této činnosti dokonce rychlejší než neinterpretované jazyky jako C. Naopak uvolňování paměti je činnost, kde v současné době interpretované jazyky zatím zaostávají. Velmi patrný je např. jev, kdy garbage collector začne uvolňovat paměť. V tomto okamžiku systém přestává reagovat a nezřídka dojde k tzv. „zamrznutí" systému. V celkovém důsledku jsou tak interpretované jazyky relativně pomalejší než neinterpretované. Tento rozdíl však není nijak dramatický a pro naše účely je tento výkonnostní deficit zcela přijatelný. Podle posledních údajů zveřejněných firmou Sun Microsystems, Inc., která je tvůrcem jazyka Java, je tento jazyk v průměru pouze asi o 10% pomalejší než neinterpretovaný jazyk C. V poslední době se objevuje celá řada projektů, které používají jazyk Java jako implementační nástroj pro tvorbu her, např. Jake 2 (předělávka PC hry Quake 2).

Další nevýhodou interpretovaných jazyků je pak paměťová náročnost, která je vyšší než u neinterpretovaných jazyků. Jedním z důvodů je například fakt, že je potřeba zajistit současně nejen běh programu, ale i virtuálního stroje. V případě Javy může být pro „běžného" uživatele také významný fakt, že virtuální stroj není v současné době součástí běžně se vyskytujících operačních systémů, jako je Microsoft Windows či Linux, ale musí být dodatečně doinstalován.

Jedním ze základních požadavků na aplikaci byla platformní nezávislost. Z tohoto důvodu by bylo jistě vhodné zvolit některý z interpretovaných jazyků Java či C#. Jazyk C# má však jednu podstatnou nevýhodu, že aplikace využívající GUI postavené na Windows Forms nejsou platformně nezávislé. V tomto případě přichází tedy v úvahu Open Sourcová implementace jazyka C# a to Mono. Mono využívá pro tvorbu GUI knihovny z projektu Gtk+. Běhové prostředí Gtk+ je přítomné dnes již snad ve všech distribucích Linuxu. Bohužel na operačním systému Windows se s ním lze setkat velmi sporadicky.

Naproti tomu jazyk Java si za svou dobu existence vydobyl velmi slušnou pozici. V podstatě lze říci, že jazyk Java zná dnes již každý, minimálně skrze aplikace pro mobilní telefony. I tento fakt zajisté přispěl ke skutečnosti, že běhové prostředí pro jazyk Java lze dnes nalézt na celé řadě počítačů. Toto běhové prostředí je stabilní součástí operačních systémů Solaris a Mac OS. Díky činnosti firmy Sun Microsystems, Inc. byla Java vstřícně přijata Open Sourcovou komunitou, což napomohlo jejímu rozšíření v rámci operačního systému Linux. I na platformě Windows je běhové prostředí pro programy napsané v jazyku Java široce rozšířeno. Tento fakt navíc podporují i výrobci počítačů, jako je HP či IBM, které toto běhové prostředí dodávají jako standardní součást výrobku.

Programovací jazyk Java patří mezi vyšší imperativní programovací jazyky s vysokým stupněm abstrakce. Tento fakt následně vede k vysoké efektivitě kódování vzhledem k ostatním jazykům. Jazyk Java patří mezi jazyky se silnou typovou kontrolou. Tato skutečnost přináší

mnohem vyšší bezpečnost do programování s ohledem na časté přetypovávání proměnných v programech. Tento rys jazyka Java je pak ještě více zvýrazněn od verze jazyka Java 1.5, kdy přibyly do jazyka generické typy. V současné době je však velmi palčivá otázka bezpečnosti nejen s ohledem na typovou kontrolu, ale hlavně otázka práce s pamětí. Při chybné práci s pamětí, ať již se jedná o alokaci, uvolňování paměti či manipulaci s ní, dochází často k chybám, které nejen ohrožují samotný program, ale v častých případech umožní útočníkovi získat nadvládu nad celým systémem. Chyby jako „buffer overflow“ a podobné jsou velmi typické např. pro jazyk C, který umožňuje programátorovi přímou manipulaci s pamětí. V případě Javy tento problém částečně odpadá, jelikož o správu paměti se stará garbage collector a programátor nemá přímý přístup k paměti. Podstatnou vlastností jazyka Java je pak kontrola hranic polí a případného přístupu pomocí indexu mimo tyto meze.

V neposlední řadě velkou výhodou Javy je intenzivní podpora ze strany programátorské komunity. Tento fakt způsobuje, že na Internetu lze bez větších obtíží nalézt celou řadu návodů a ukázkových řešení. Důležitý je však také přístup ze strany samotného tvůrce jazyka, firmy Sun Microsystems, Inc. Autoři se snaží nejen poskytnout kvalitní dokumentaci, ale i celou řadu návodů. Dále je také velmi podstatný fakt, že se firma Sun Microsystems, Inc. snaží flexibilně reagovat na případné bezpečností problémy samotného jazyka skrze časté uvolňování nových verzí běhového prostředí JRE (Java Runtime Environment).

Osobně si myslím, že programovací jazyk Java je velmi vhodný pro implementaci této bakalářské práce z několika důvodů. Tím hlavním je asi multiplatformnost. Dalším podstatným důvodem je zcela jistě typová kontrola při práci s objekty a bezpečnost při práci s pamětí. V neposlední řadě je to i vysoká efektivita programování. Jazyk Java má sice řadu nevýhod, např. rychlost běhu aplikací či paměťové nároky, ale domnívám se, že klady tohoto jazyka hovoří dostatečně přesvědčivě pro jeho volbu.

### 5.1.2 Volba databázového serveru

Na trhu s databázovými servery lze nalézt celou řadu produktů od společností jako je Oracle, Microsoft atd. Tyto produkty jsou velmi kvalitní, ale mají také řadu nevýhod. Mezi jejich hlavní nevýhody patří vysoké náklady na pořízení a často bývají svázány pouze s určitou platformou.

Zajímavou alternativou k výše zmíněným produktům může být databázový server MySQL od firmy TcX. V tomto případě se jedná o vysoce výkonný relační databázový server, který umožňuje pojmout velké množství dat, aniž by došlo k nějakému znatelnému snížení výkonu. Podstatnou výhodou je fakt, že není svázaný s žádnou platformou, a proto ho lze nasadit na všech doposud známých operačních systémech. Velmi příjemnou skutečností je, že MySQL server je šířen jako Open Source projekt, a proto ho lze pro nekomerční účely využívat zdarma. V případě komerčního využití je nutné zaplatit určité licenční poplatky viz. www.myqsl.com. Naštěstí i v případě komerčního využití se jedná o velmi nízké poplatky s ohledem na konkurenční řešení od ostatních firem. Pokud si však zakoupíte licenci, získáte tak nejen přístup ke zdrojovým kódům, ale hlavně máte právo legálně tyto zdrojové kódy upravovat. Výrobce uvádí, že lze vlastními úpravami dosáhnout až 40% nárůstu výkonu. Tak lze docílit požadovaného chování serveru i ve specifických podmínkách nasazení. V případě nekomerčního využití máte právo pouze nahlédnou do zdrojových kódů za účelem studia.

Stručné shrnutí výhod databázového serveru MySQL:

1. Platformní nezávislost a snadný přechod mezi platformami.
2. Vysoký výkon oproti ostatním Open Source produktům.
3. Možnost pojmout velké množství dat bez znatelné ztráty výkonu.
4. Pro nekomerční účely lze používat zdarma.

5. V případě komerčního využití možnost zásahu do zdrojových kódů.

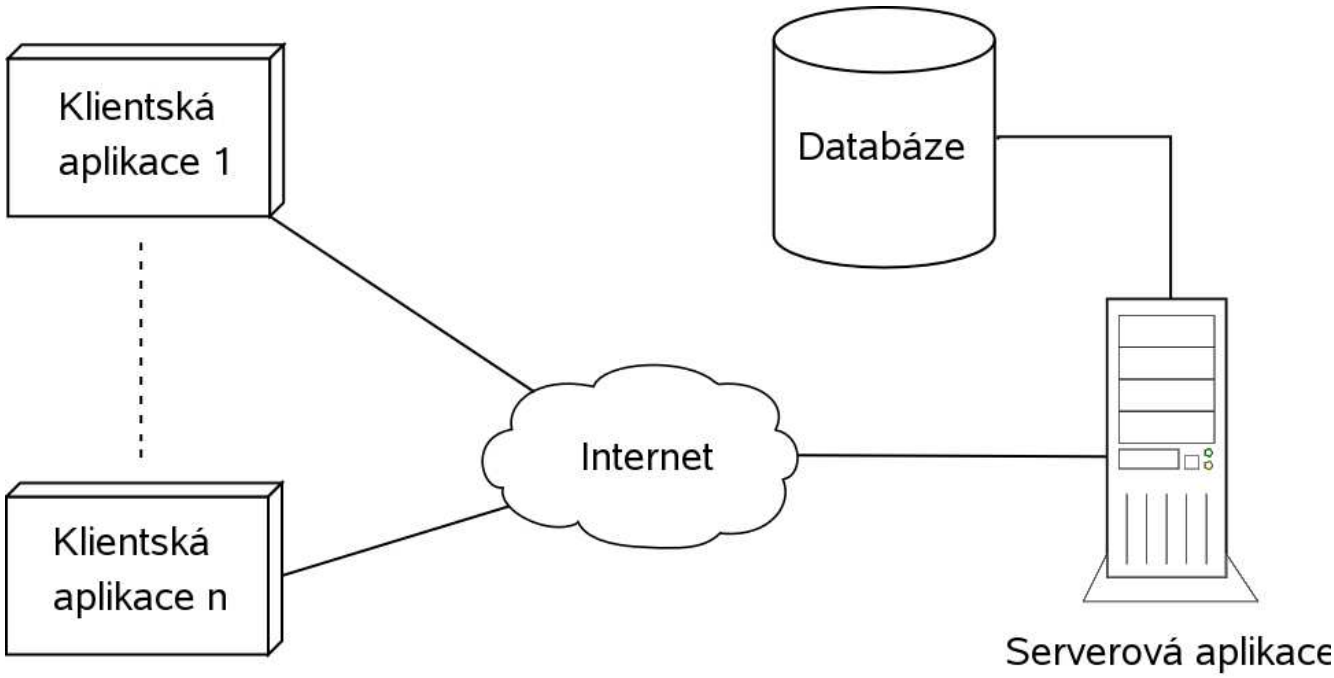
Tento produkt zcela jistě najde uplatnění všude, kde je potřeba výkonný databázový systém. V současné době se s tímto produktem lze nejčastěji setkat jako databázové řešení pro webové servery. Tato skutečnost je zcela logická, jelikož každá aplikace potřebuje úložiště dat, které bude rychlé, spolehlivé, dokáže pojmout velké množství dat bez znatelné ztráty výkonu a to vše za velmi příznivou cenu. Všechny výše zmíněné požadavky na úložiště dat MySQL databázový server zcela splňuje. V této oblasti si tento produkt již vydobyl významnou pozici. Zároveň je však třeba říci, že tento produkt není určen pouze pro webové servery. Lze velmi snadno nalézt jeho uplatnění i v podnikových sítích a různých intranetech. I v těchto případech plní roli úložiště dat.

V dnešní době každá firma kromě papírových záznamů vytváří i celou řadu dokumentů v elektronické podobě. Ať již jsou tyto údaje určeny k soukromým účelům firmy nebo k zveřejnění, vždy je potřeba tyto dokumenty zabezpečit proti ztrátě. A právě tuto schopnost databázový server MySQL poskytuje. MySQL server je schopen zajistit nejen případnou ochranu před ztrátou dokumentů, ale také před jejich zneužitím. K takto uloženým datům lze následně přistupovat skrze celou řadu rozhraní. V současné době je k dispozici rozhraní pro Javu, C, C++, C#, PHP atd.

Po dlouhé úvaze jsem se rozhodl zvolit databázový server MySQL 5.0. Jedním z hlavních důvodů byla cena tohoto serveru, která je v tomto případě nulová. Dále je třeba určitě zmínit skutečnost, že se jedná pravděpodobně o nejvíce rozšířený databázový server, ke kterému je přístup zdarma. Z tohoto faktu proto plyne, že na Internetu lze nalézt velké množství návodů pro práci s tímto serverem. Určitě je také podstatný fakt, že se jedná v dané kategorii o vysoce výkonné a spolehlivé řešení databáze. Mezi kladné vlastnosti určitě také patří snadná správa a konfigurace tohoto serveru. Výrobce poskytuje zdarma kvalitní dokumentaci a příjemný prohlížeč záznamů v databázi. Důležitá je také podpora jazyka Java samotným výrobcem, který poskytuje zdarma konektor JDBC do databáze.

## 5.2 Návrh architektury aplikace

Protože se jedná o aplikaci, která má umožňovat současnou práci více uživatelů byla zvolena architektura klient-server. Obrázek 3 znázorňuje architekturu celého systému. Dále má program umožňovat, prohlížet si plány ostatních uživatelů na základě uživatelských práv. Z tohoto důvodu byl za úložiště dat zvolen relační databázový server. Klientská část je tvořena klientskou aplikací, kterou má uživatel na svém lokálním disku. Klientská aplikace při spuštění naváže kontakt se serverovou částí. Po následné autentizaci klientská aplikace zajišťuje komunikaci se serverovou částí. Díky faktu že klientská aplikace je na lokálním disku klienta, lze uložit inicializační údaje aplikace na lokální disk uživatele a není nutné proto tyto údaje přenášet po sítí při každém spuštění klientské aplikace. Inicializační data obsahují např. nastavení barevného schéma aplikace.




**Obrázek 3: Architektura aplikace MUSC**

Serverová část je tvořena serverovou aplikací. Server se stará o příjem požadavků od klientské části a o jejich následné zpracování. Některé požadavky může obsloužit přímo sám server, jiné vyžadují získání dat z databáze. Pokud je nutná interakce s databází, tak server vytvoří spojení do databáze a vykoná příslušný dotaz. Jak je patrné, klientská aplikace nemá přímý přístup k datům v databázi. Získání dat z databáze se děje zprostředkovaně skrze serverovou část aplikace. Tato skutečnost přispívá ke zvýšení bezpečnosti celého systému. I na straně serveru je soubor s inicializačními údaji, který obsahuje např. informace pro připojení do databáze.

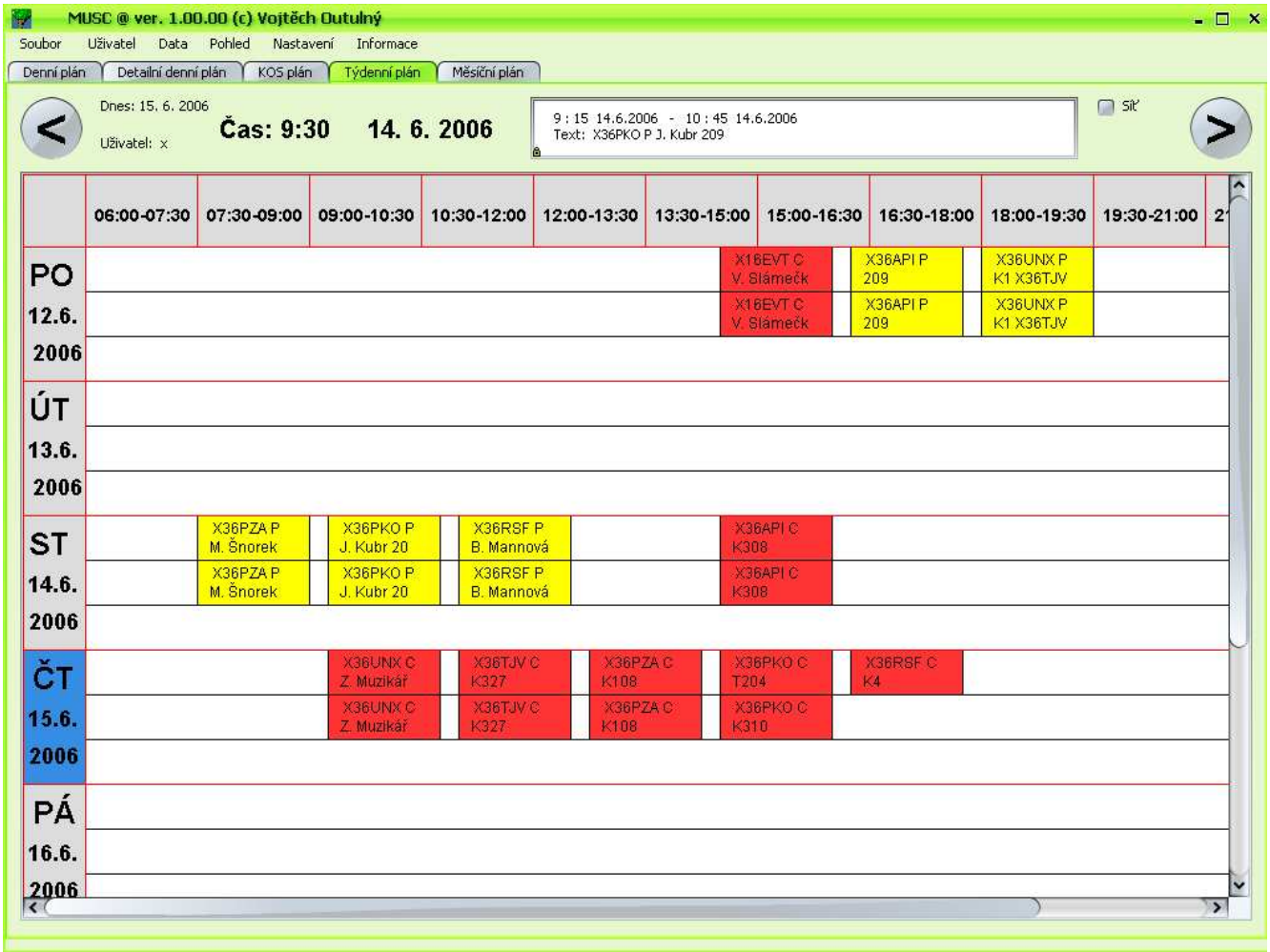
Jako úložiště dat byl zvolen databázový server. Databázový server zvyšuje bezpečnost proti nepovolanému přístupu k datům ostatních uživatelů. Dále databáze nabízí účinnou ochranu proti možné ztrátě dat. V neposlední řadě poskytuje i efektivní přístup k datům v ní uložených.

## 5.3 Návrh uživatelského prostředí

Na uživatelské prostředí je kladena celá řada požadavků, jako je přehlednost, intuitivnost, možnost snadné a rychlé práce. Také je třeba zajistit, aby GUI na uživatele působilo příjemným a pozitivním dojmem. Zároveň je však také důležité, aby vzhled nijak neomezoval uživatele při práci a poskytl mu veškeré potřebné informace. Návrh GUI klientské aplikace je vidět na obrázku 4.

V dnešní době většina aplikací po vzhledové stránce trpí jednotným vzhledem, který je dán „Look And Feelem“ operačního systému. Další aplikace s podobným vzhledem by však uživatele pravděpodobně neuchvátila, a proto jsem se rozhodl využit pro aplikaci „Look And Feel“ Substance, jehož autorem je Kirill Grouchnikov. Tento „Look And Fell“ umožňuje nastavit celou řadu vlastností pro zobrazované komponenty. Již samotný rám okna je řešen tak, aby společně se zbytkem celé aplikace vytvářel jednotné prostředí, na rozdíl od typických situací u dnešních aplikací, kdy Window manager operačního systému definuje vlastní rám a uživatel si nezávisle na něm definuje zbytek aplikace, což ve výsledku působí nesourodým dojmem. Tento „Look And Feel“ dále umožňuje volit např. tvary tlačítek a jejich podklad. Zároveň poskytuje možnost definovat pozadí aplikace. Všechny tyto možnosti pak lze zabarvit do celé řady barevných odstínů.

Každý uživatel má specifické požadavky na barevné ztvárnění aplikace, ať už se jedná o pozadí nebo barvy pro jednotlivé typy priorit. Z tohoto důvodu bude uživateli poskytnuta možnost nastavení barevného schéma aplikace dle jeho požadavků.




**Obrázek 4: Návrh GUI klientské aplikace**

Aplikace bude rozdělena do dvou částí. První částí je menu, které poskytuje řadu funkcí, jež nejsou přímo nutné pro prohlížení, tvorbu a editaci plánů. Tyto funkce slouží pro specifické situace při práci s plány. Druhou částí je pak záložkový panel, který slouží pro snadnou orientaci a rychlý přechod mezi pohledy.

Samotná tvorba a editace plánu by měla být co nejvíce intuitivní a zároveň by měla poskytovat komfortní a rychlé prostředky pro tyto činnosti. Asi nejpřirozenějším způsobem je kliknout v místě, kde má akce začínat, a tahem myši určit její konec. Program by tak byl schopen přibližně určit časové období pro danou akci a uživatel by se již nemusel zabývat specifikací těchto časů. Protože uživatel většinou jen přibližně určí trvání akce, je potřeba mu dát možnost specifikovat tyto informace přesněji. Z tohoto důvodu by se po uvolnění myši otevřel dialog. Aby byla tvorba co nejintuitivnější a nejrychlejší, neměl by dialog nabízet přemíru možností. V tomto dialogu by mohl uživatel určit text akce, případně upřesnit čas akce. Někteří uživatelé však chtějí přesněji specifikovat vytvářené akce, a tak by dialog měl umožňovat rychlé zobrazení všech dalších možných nastavení.

Výběr akcí pro editaci a mazání by byl řešen pomocí jednoduchého kliknutí na požadovanou akci. Poté se zobrazí podobný dialog jako pro tvorbu akcí, který však umožňuje editaci a mazání akcí.

Z rešerše vyplynulo, že pravidlem pro podobné programy je poskytnutí minimálně tří základních pohledů – Denní plán, Týdenní plán a Měsíční plán. Tento výběr však může být v celé řadě případů nedostačující. Např. mnoho pracujících lidí zajímá co se děje pouze v rámci pracovního týdne od pondělí do pátku a nechtějí ho míchat s víkendem a volným časem. Existuje i řada lidí, kteří si chtějí svůj čas naplánovat velmi přesně. Z těchto důvodů jsem se rozhodl pro tvorbu pěti rozdílných pohledů – Denní plán, Detailní denní plán, KOS plán, Týdenní plán a Měsíční plán.

Denní plán bude umožňovat zobrazení akcí v rámci jednoho dne. Zároveň bude umožňovat zobrazení více souběžně probíhajících akcí. Naproti tomu Detailní denní plán bude umožňovat co nejpřesnější plánování v rámci jednoho dne. Zároveň umožní zobrazit extrémně velké množství souběžně probíhajících akcí, aby si uživatel mohl naplánovat „opravdu cokoliv" a nebyl nijak omezován. Třetí pohled by umožňoval zobrazit pracovní týden. S ohledem na to že mnoho lidí je zvyklá využívat rozvrh z KOSu, byl by tento pohled koncipován podobným způsobem. Zároveň by mohl být vylepšen např. o zobrazení přestávek mezi jednotlivými vyučujícími hodinami. Dalším pohledem je Týdenní plán umožňující zobrazení klasického týdne od pondělí do neděle. Pátým a zároveň posledním pohledem bude Měsíční plán. V MS Outlooku 2000 jsou v měsíčním pohledu zobrazeny pouze jednotlivé akce z jednotlivých dní bez nějakého spojení s plánováním po delších časových úsecích. Mesíční plán u MUSC by se zaměřil spíše na plánování delších akcí, které se odehrávají více dní. T tohoto důvodu by zde byla pouze informace o existenci jednodenních akcí a zobrazovány by byly pouze vícedenní akce. Díky tomu bude zajištěna možnost plánovat „opravdu po měsících".

## 5.4 Návrh datových formátů pro uložení plánu a nastavení aplikace

Jedním z požadavků na program MUSC, bylo umožnění tvorby plánu více rozdílnými způsoby. V programu MUSC budou existovat v zásadě tři varianty tvorby a vkládání plánu do aplikace. První možností je tvorba akcí skrze GUI klientské aplikace programu MUSC. Zde lze velmi jednoduchým způsobem pomocí tažení myší vytvořit akce. Druhou možností vložení dat je import rozvrhu z KOSu v HTML souboru. Třetí možností jak tvořit akce je pak možnost vložení akcí do aplikace skrze funkci „Importovat plán".

Akce musí být uloženy na lokálním disku klienta v souboru obsahující akce, které jsou ve vlastním formátu aplikace. Následně je uživatel vyzván pro zadání pozice souboru na disku. Po udání polohy je proveden import akcí do programu MUSC. Akce jsou dočasně uloženy v lokální vyrovnávací paměti. Importované akce jsou natrvalo uloženy až při provedení volby „Uložit plán". Pokud by uživatel ukončil program aniž by provedl uložení plánu, budou importovaná data nenávratně ztracena a při dalším spuštění je třeba provést import akcí znovu.

Na datový formát byla kladena řada požadavků. Tento datový formát měl umožňovat vložení akcí do programu. Dále měl tento datový formát poskytovat možnost exportovat akce a následně soubor v tomto formátu použít pro přenos akcí mezi více uživateli. Dále měl formát také umožňovat snadnou a rychlou tvorbu akcí bez nutnosti použití klientské aplikace programu MUSC. Tento požadavek nutně vedl k tomu, aby struktura souboru byla co nejpřehlednější a aby uživatel vždy věděl jak akci vytvořit. Další požadavek byl také umožnit snadnou a rychlou tvorbu opakujících se akcí bez nutnosti tvorby všech akcí ze strany uživatele. Častým problémem u takovýchto editovatelných souborů pak bývá skutečnost, že uživatel vytvoří akci špatně. Tato skutečnost může být např. způsobena nezadáním některých informací nebo překlepy atp. Z tohoto důvodu je proto nutná syntaktická kontrola tohoto dokumentu ještě před samotnou tvorbou akcí. Nelze však také opomenout negativní úmysly uživatele pokusit se aplikaci poškodit zadáním např. příliš dlouhých vstupních údajů atp. Proto i když struktura souboru je po syntaktické stránce správná, je třeba provést ještě kontrolu všech vstupních dat.

Program MUSC bude také umožňovat uživateli nastavit si aplikaci podle svého přání. Nastavit lze barvu pozadí a záhlaví pohledů atd. Dále si uživatel může také nastavit např. barvy pro zobrazení jednotlivých priorit. Protože si uživatel bude chtít své nastavení trvale uchovat, je třeba toto nastavení uložit do externího souboru. Datový formát musí proto umožňovat snadno upravit uživatelovo nastavení.

Výše nastíněné požadavky na datové formáty nutně vedou ke vzniku komplikovaných a sofistikovaných formátů. V zásadě byly dvě možnosti vyřešení tohoto problému. První variantou bylo vytvoření vlastních datových formátů. Zároveň však bylo třeba vytvořit další podpůrné nástroje, jako je modul pro import akcí nebo modul pro načtení nastavení v těchto formátech, dále pak vytvořit validátor pro syntaktickou kontrolu atp. Druhou variantou bylo vytvořit pouze gramatiku pro tyto datové formáty a následně využit již existujících podpůrných nástrojů. Z těchto dvou variant jsem se rozhodl pro druhou a to pro definování gramatiky těchto datových formátů.

Následně nastala otázka, jaké lze využít podpůrné nástroje. V tomto případě bylo potřeba vzít v potaz několik požadavků. Jedním z nich je aby tyto nástroje nekladly na uživatele žádné další nároky, jako je např. instalace dalších podpůrných programů. Dalším požadavkem byla také nezávislost těchto nástrojů na použité platformě. Dále je třeba zajistit, aby tyto nástroje byly schopny vzájemně spolupracovat bez problémů. Vhodné by bylo, kdyby podpůrné prostředky a nástroje pro definici datového formátu byly součástí jednoho celku, který je zároveň standardizovaný některou standardizační organizací. Tato skutečnost by zaručoval, že soubory, ač ve vlastním datovém formátu aplikace, by mohly být v budoucnu přenositelné mezi programy různých výrobců a mezi více platformami. Výše zmíněné důvody vedli k volbě realizovat datový formát pomocí XML a vhodného nástroje pro definici gramatiky.

### 5.4.1 Schéma pro definici dokumentu

Starší formát pro schéma je DTD (Document Type Definition), který vychází z SGML (Standard Generalized Markup Language). DTD se používá pro popis dokumentů. DTD primárně

používá pro vyjádření gramatiky množinu deklarací, které se přizpůsobují částečně značkové syntaxi a popisují třídu nebo typ SGML XML dokumentu. Ačkoliv DTD je přímo součástí standardu XML 1.0 je zde několik důvodu, kvůli nimž je DTD limitováno:

1. Není zde podpora pro nové vlastnost XML standartu, jako jsou např. jmenné prostory.

2. Nedostatek prostředků pro vyjadřování. Některé formální aspekty XML dokumentů nelze pomocí DTD zachytit.

3. K popisu gramatiky schématu nepoužívá jazyk XML, ale místo toho využívá syntaktický zápis z SGML.

4. Nepodporuje základní datové typy jako je integer nebo string.

Novější jazyk pro popis gramatiky je XML Schéma, které má být následníkem DTD. XML schéma bylo uvedeno organizací W3C v květnu 2001. XML Schéma může být použito pro definici gramatiky XML dokumentů. XML Schéma obsahuje řadu pravidel, kterým se musí dokument podřídit, aby byl shledán „validním" z pohledu daného schématu. Obvykle se místo označení XML Schéma používá název XSD (XML Schema Definition). XSD je mnohem mocnější v popisu XML jazyku než DTD. XSD používá bohatý systém různých datových typů, který umožňuje detailněji určit logickou strukturu XML dokumentu. Z tohoto důvodu je však třeba používat mnohem robustnější nástroje pro validaci dokumentů. Výhodou u XSD je, že je založeno na XML formátu, a proto využívá syntaxi XML.

Jak již bylo zmíněno, je v současné době třeba klást velký důraz na bezpečnost. Z tohoto důvodu je třeba dbát, aby dokument byl vždy validní. Je proto nutné, aby byla zajištěna kontrola vstupních dat, např. na místě kde se očekává číslo se musí opravdu vyskytovat číslo a ne text. Je zde třeba zdůraznit, že se nejedná pouze o validní z pohledu syntaxe, ale také z pohledu sémantiky. Z tohoto důvodu je potřeba např. také zajistit, aby u číselných údajů nedošlo k překročení rozsahů stanovených aplikací pro dané hodnoty. Další podstatnou výhodou u XSD je fakt, že využívá syntaxi XML. Tato syntaxe je velmi přehledná. Přehlednost a snadná čitelnost dokumentů vede ke skutečnosti, že uživatel může snadno vytvářet XML dokument podle jasné a předem určené struktury. Navíc všechny značky v XML dokumentu lze logicky pojmenovat a uživatel, proto vždy ví co má kam napsat. Jedním z důvodů pro výběr XML a XSD je také skutečnost, že XML a XSD je přímo podporováno jazykem Java. Ze všech výše zmíněných důvodů jsem se nakonec rozhodl pro definování gramatiky skrze XSD a jazyk XML použít pro uložení dat.

# 6 Implementační část

## 6.1 Popis jednotlivých tříd

Zdrojové kódy jsou rozděleny do šesti balíčků – musc.client (45 tříd), musc.commands (19 tříd), musc.security (3 třídy), musc.server (1 třída), musc.server.socket (1 třída) a musc.server.database (2 třídy). Přibližně lze říci, že program je tvořen asi 71 třídami, což je ale pouze přibližné číslo, protože ve zdrojových kódech se vyskytuje ještě celá řada vnitřních tříd. Jelikož počet tříd je příliš velký na to, aby zde byly popsány všechny třídy, vyberu si k jednoduchému popisu pouze několik základních tříd.

### 6.1.1 Balíček musc.client

Balíček musc.client poskytuje tři typy tříd. Prvním typem jsou třídy pro zajištění komunikace mezi klientem a server. Druhým typem jsou třídy pro vytvoření uživatelského rozhraní a zobrazení dat získaných z datového modelu. Posledním typem tříd jsou třídy zajišťující datový model a manipulaci s ním.

**Třída DataBuffer**

Třída DataBuffer implementuje rozhraní Serializable. Tato třída slouží k dočasnému uchovávání dat získaných z databáze a dat připravených k uložení do databáze. Třída obsahuje osm vyrovnávacích pamětí pro ukládání dat, které jsou tvořeny statickými proměnnými typu java.util.ArrayList. Proměnná buffer slouží k uchovávání dat získaných z databáze a dat určených k zobrazení v jednotlivých pohledech. Buffer bufferNewAction obsahuje nově vytvořené akce, které je třeba teprve uložit do databáze. Buffer bufferDelete obsahuje akce určené k smazání a buffer bufferUpDate obsahuje akce, které je potřeba modifikovat v databázi. Buffer bufferDeleteRepeat obsahuje opakující se akce určené k smazání a buffer bufferUpDateRepeat obsahuje opakující se akce, které je potřeba modifikovat v databázi. Buffer bufferDeleteRepeatKOS obsahuje akce z rozvrhu určené ke smazání a buffer bufferUpDateRepeatKOS obsahuje akce z rozvrhu, které je potřeba modifikovat v databázi. Protože data z databáze jsou načítána pouze pro určité časové období (obvykle aktuální, předešlý a nadcházející měsíc), třída obsahuje proměnné obsahující informaci o datu a času krajních akcí z právě načteného časového intervalu. Třída DataBuffer také poskytuje statickou metodu seradDataBuffer() pro setřídění dat v rámci bufferu. Dále třída poskytuje celou řadu metod pro úpravu jednotlivých bufferů v případě tvorby, editace a mazání akcí.

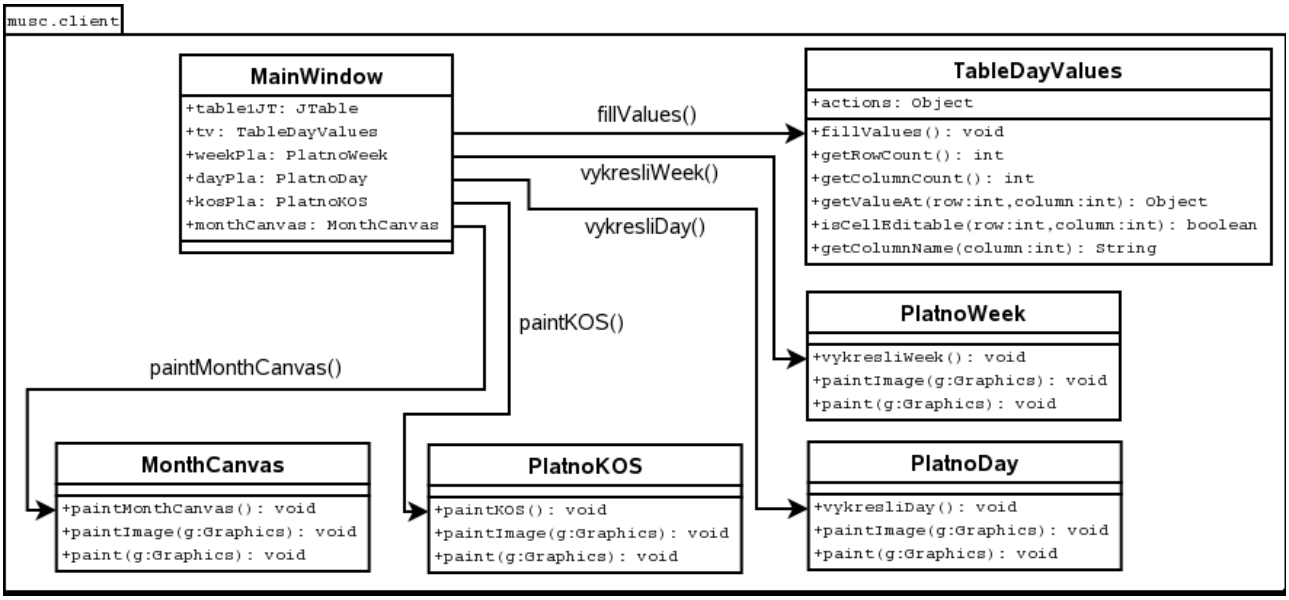
**Třída ClassClient**

Třída ClassClient slouží k zajištění komunikace mezi klientskou aplikací a serverou aplikací. Třída obsahuje proměnou array typu java.util.ArrayList, která slouží k ukládání příkazů, které mají být odeslány na server. Pomocí metody addCommand(Command c) lze přidat další příkaz k odeslání. Statická metoda setParametersConnection(String h,int p) slouží k nastavení cílové adresy serveru a portu na němž server poslouchá. Metoda createConnection() slouží k navázání spojení se serverem. Tato metoda vrací objekt typu Command. Návratová hodnota je buď typu null, když se nelze spojit se serverem, nebo typu ReturnData, která nese informaci o návratovém kódu operace provedené na serveru a případně i získaná data ze serveru.

**Třída MainWindow**

Tato třída dědí od třídy javax.swing.JFrame. Jak je zřejmě patrné z názvu, tato třída se stará o zobrazení GUI (Graphical User Intergace). Celé GUI je vlastně rozděleno na dvě části.

První částí je menu a druhou záložkový panel. Menu je instance třídy MyMenu. Záložkový panel slouží jako kontajner pro jednotlivé pohledy, které jsou tvořeny opět kontajnerem typu JPanel pro jednotlivé prvky pohledů. Do první záložky je vložena instance třídy PlatnoDay, do druhé JTable, do třetí PlatnoKOS, do čtvrté PlatnoWeek a do čtvrté MonthCanvas, které slouží pro zobrazení dat v různých pohledech. Obrázek 5 zobrazuje komunikaci mezi třídou MainWindow a třídami pro vykreslení jednotlivých pohledů. Dále třída MainWindow obsahuje celou řadu vnitřních tříd, které slouží jako posluchače událostí pro jednotlivé komponenty GUI.




**Obrázek 5: Komunikace mezi třídou MainWindow a třídami pro jednotlivé pohledy**

### Třída MyMenu

Třída MyMenu dědí od třídy javax.swing.JMenuBar a slouží k vytvoření celého menu. JMenuBar obsahuje šest instancí třídy JMenu (fileJM, userJM, editJM, viewJM, optionsJM a infoJM), které vytváří samotné menu. Menu fileJM slouží k zajištění základních činností jako je uložení plánu, konec atd. Menu userJM nabízí služby pro práci s uživatelem jako je přihlášení nebo tvorba nového účtu. Dále menu editJM poskytuje možnost importu a exportu plánů z/do aplikace. Menu viewJM poskytuje např. náhled na vybranou akci nebo informace o uživateli atd. Menu optionsJM poskytuje celou řadu nastavení barevného schéma aplikace. Nakonec menu infoJM poskytuje základní informace o aplikaci.

### Třída TableDayValues

Tato třída dědí od třídy AbstractTableModel a vytváří datový model pro tabulku, která reprezentuje Detailní denní plán. Třída obsahuje dvourozměrné pole jménem values typu Object, které slouží k uchování dat, která jsou následně zobrazena tabulkou table1JT v třídě MainWindow. Důležitou metodou je zde metoda fillValues(), která slouží k naplnění matice values daty, pokaždé když dojde ke změně dat získaných z databáze. Za povšimnutí stojí určitě vnitřní třída TimeStruct, která reprezentuje časové linie pro souběžně probíhající akce. Tato třída si uchovává informaci o jakou jde časovou linii (číslo sloupečku v matici), číslo řádku v matici (kde skončila poslední zobrazovaná akce pro danou časovou linii) a koncový čas poslední zobrazené akce v rámci dané časové linie.

### Třída MyTableCellEditor

Tato třída dědí od třídy AbstractCellEditor a implementuje rozhraní TableCellEditor.

Následně implementuje metodu getTableCellEditorComponent(JTable table,Object value,boolean isSelected,int row,int column), která slouží k navrácení komponenty pro editaci jednotlivých buněk v tabulce. Pro editaci jsem zvolil komponentu java.swing.JLabel, která po kliknutí na ní vyvolá příslušný dialog pro tvorbu nové akce nebo případně pro editaci či smazání již existující akce.

**Třída MyTableCellRenderer**

Třída MyTableCellRenderer dědí od třídy javax.swing.JLabel a implementuje rozhraní TableCellRenderer. Komponenta JLabel je použita pro zobrazení dat obsažených v proměnné values, která je součástí třídy TableDayValues.

**Třídy jednotlivých pohledů**

Pro vykreslování plánu slouží třídy PlatnoDay, PlatnoKOS, PlatnoWeek a MonthCanvas. Třída PlatnoDay slouží k vykreslení jednoho dne v rámci Denního plánu a umožňuje zobrazení až deseti současně probíhajících akcí. Pro vykreslování slouží metoda vykresliDay(). Třída PlatnoKOS slouží k vykreslení pracovního týdne v KOS plánu. Zároveň tento pohled umožňuje zobrazení až tří paralelně probíhajících akcí. Pro vykreslení je použita metoda paintKOS(). Další třídou je PlatnoWeek, která slouží k vykreslení akcí v Týdenním plánu od pondělí do neděle a umožňuje zobrazení maximálně tří souběžných akcí. Vykreslování probíhá pomocí metody vykreliWeek(). Poslední třídou je třída MonthCanvas, která slouží k vykreslování Měsíčního plánu. Každý den je rozdělen na dvě samostatné části. V dolní části je indikátor informující, jestli se v daný den vyskytují nějaké akce, které se odehrávají pouze v rámci daného dne. V horní části jsou zobrazeny akce, které trvají déle než jeden den. Měsíční pohled umožňuje zobrazení až dvou současně probíhajících vícedenních událostí. Více o algoritmu pro vykreslování dat v pohledech se dozvíte v kapitole 6.5.1 Popis vykreslování dat v jednotlivých pohledech.

**Třída ColorOptions**

Tato třída slouží k změně a ukládání barevného schéma aplikace. Pomocí této třídy lze např. změnit barvu pozadí nebo záhlaví v jednotlivých pohledech. Pomocí metody setColorPriority(int i,Color c) lze přiřadit jednotlivým prioritám, které jsou určeny celým číslem v intervalu od nuly do jedenácti, barvu pro zobrazení. Tato třída také nabízí skrze metodu loadUserOptions() načtení uživatelem uloženého nastavení ze souboru MUSCoptions.xml. Dále lze pomocí metody setDefaultColors() nastavit původní nastavení aplikace.

### 6.1.2 Balíček musc.security

Tento balíček poskytuje třídy pro zabezpečení komunikace mezi klientem a serverem pomocí šifrování.

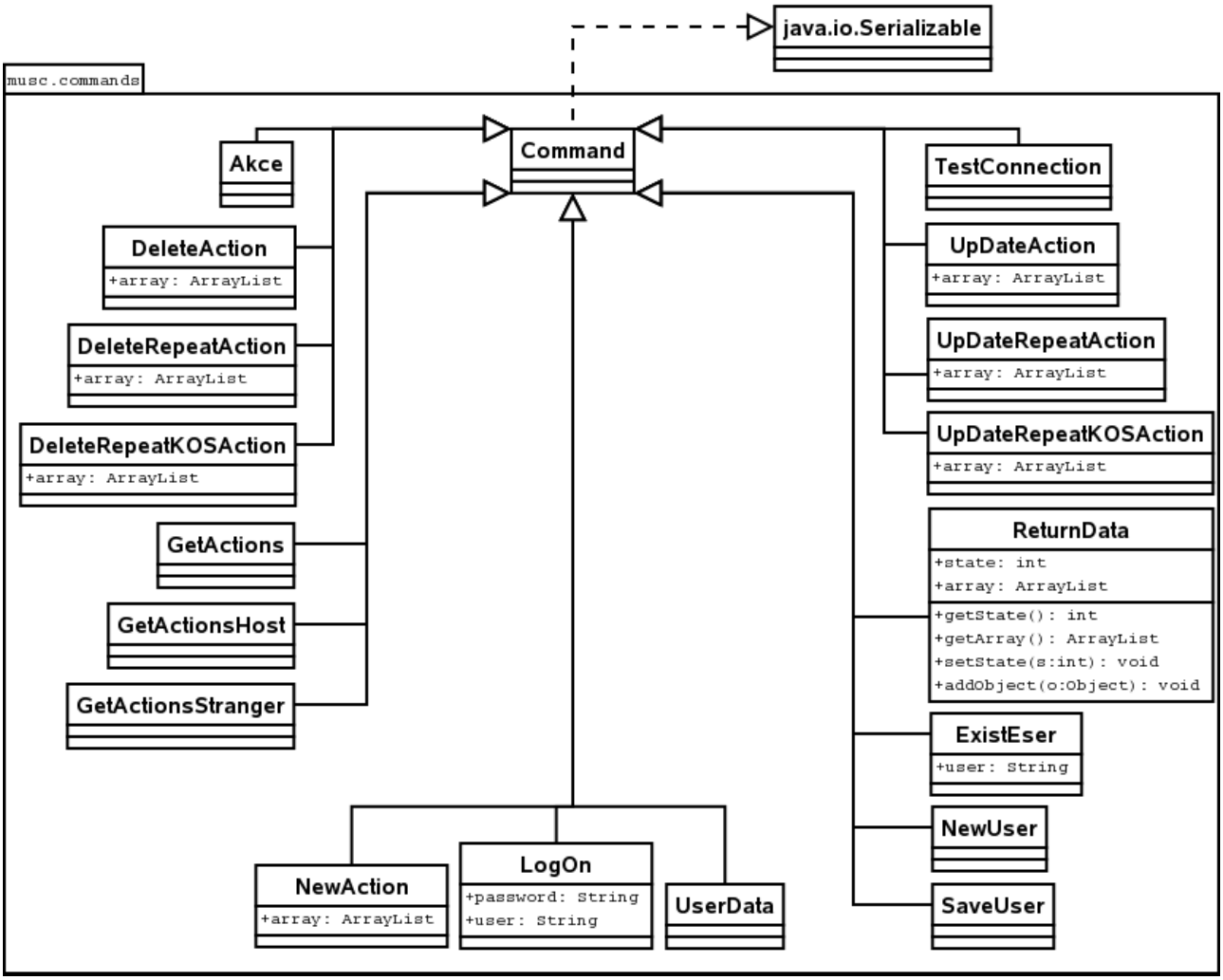
**Třída Autenticity**

Tato třída slouží ke dvěma účelům. Prvním je načtení identifikačních údajů z disku uživatele pomocí metody loadAuthenticity(String file,Key key), kde argument file udává cestu k souboru s údaji a argument key slouží jako klíč k dešifrování. Jelikož jsou data na disku uložena v zašifrované podobě, je třeba je po načtení dešifrovat. Následně načtené uživatelské jméno a heslo jsou uloženy do instančních proměnných userName a password.

Druhým účelem je pak ukládání identifikačních údajů na lokální disk klienta, což se děje skrze metodu saveAuthenticity(String file,Key key). Agument file určuje název a umístění souboru do něhož budou údaje uloženy. Z důvodu bezpečnosti jsou data zašifrována algoritmem Blowfish s využitím klíče key.

### 6.1.3 Balíček musc.commands

Tento balíček poskytuje třídy jejichž instance reprezentují příkazy s požadavky zasílané klientem na server. Podrobnější popis tříd obsažených v balíčku musc.commands lze nalézt v rámci kapitoly 6.2 Komunikace mezi klientem a serverem. Obrázek 6 ukazuje přehled tříd reprezentujících jednotlivé příkazy v balíčku musc.commands.



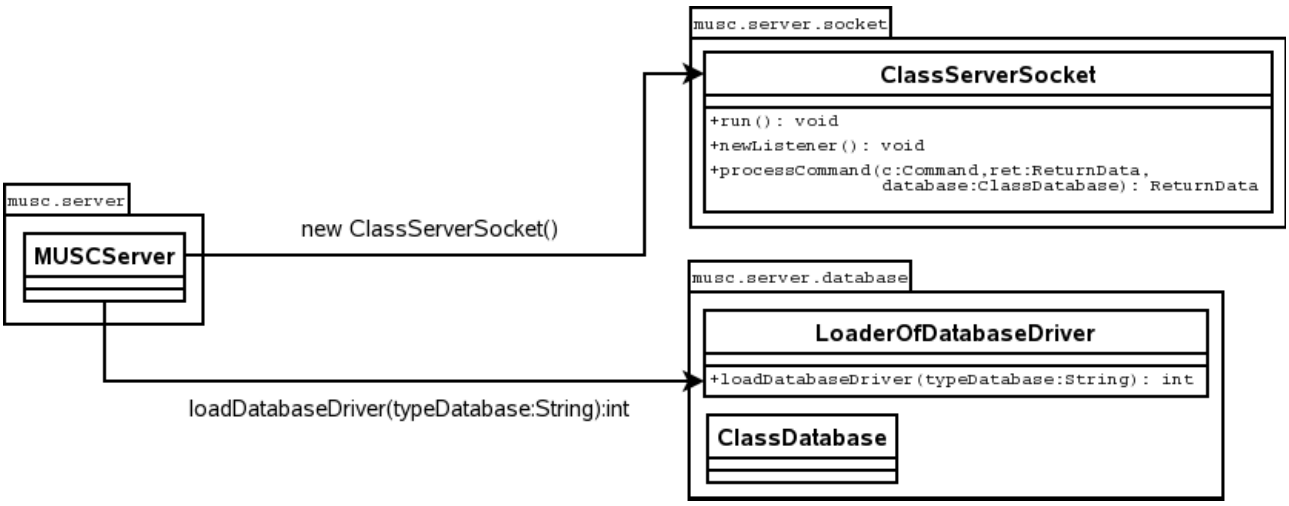

**Obrázek 6: Přehled tříd reprezentujících jednotlivé příkazy**

### 6.1.4 Balíček musc.server

Tento balíček poskytuje třídy pro zpracování vstupních parametrů pro server a databázi. Také zajišťuje spuštění serveru bez ohledu na následně použitou technologii pro komunikaci mezi klientem a serverem.

#### Třída MUSCServer

Třída MUSCServer slouží především pro inicializaci serveru. Nejprve provede zpracování vstupních argumentů programu a poté podle vybraného typu databáze registruje příslušný ovladač pro danou databázi. Nakonec spustí samotný server. Na obrázku 7 lze vidět, jak probíhá inicializace a spuštění serveru.




**Obrázek 7: Inicializace a spuštění serveru**

### 6.1.5 Balíček musc.server.socket

Balíček musc.server.socket poskytuje třídy pro zajištění komunikace mezi klientem a serverem s využitím konkrétní technologie socketů. Dále zajišťuje zpracování požadavků klienta na server.

#### Třída ClassServerSocket

Třída ClassServerSocket implementuje rozhraní Runnable a zajišťuje základní funkcionalitu serveru, komunikaci s klientem a zpracování jeho požadavků. V rámci svého vytvoření a inicializace dojde k připravení šifer využívaných během komunikace a k otestování spojení do databáze. Metoda newListener() slouží k vytvoření nového vlákna, aby byla zaručena možnost souběžného zpracování více požadavků od různých klientů. Je zde implementována metoda run() z rozhraní Runnable, která je vykonávána každým nově vytvořeným vláknem. V rámci této metody server poslouchá na daném portu a pokud přijde požadavek na komunikaci, tak tento požadavek akceptuje. Poté dojde k vytvoření nového vlákna pro obsluhu dalších klientů. Následně je získán vstupní stream z přijatého spojení, z kterého jsou poté vyčteny požadavky na činnost serveru. Po dešifrování jsou tyto požadavky předány metodě processCommand(Command c,ReturnData ret, ClassDatabase database), která příslušné požadavky zpracuje. Po zpracování všech požadavků dochází k zašifrování dat získaných během operace a následně jsou takto zabezpečená data odeslána klientovi.

Metoda processCommand(Command c,ReturnData ret, ClassDatabase database) slouží ke zpracování požadavků na server. Nejprve je identifikováno o jaký se jedná příkaz a poté je příkaz vykonán. Pokud je v požadavku vyžadována interakce s databází, je k tomuto účelu využit parametr database. Parametr ret slouží k uložení výsledků operace.

### 6.1.6 Balíček musc.server.database

Balíček musc.server.database poskytuje třídy pro práci s databází. To může být nejen vytvoření spojení s databází a provedení požadovaných akcí, ale také registrace ovladač pro příslušný typ databáze.

**Třída LoaderOfDatabaseDriver**

Třída LoaderOfDatabaseDriver slouží k registraci ovladače pro komunikaci s databází. K tomuto účelu slouží statická metoda loadDatabaseDriver(String typeDatabase). Vstupní parametr typeDatabase určuje typ databáze pro níž má být ovladač zaregistrován. Návratovou hodnotou je pak proměnná typu integer která určuje stav, zda se operace povedla (1) nebo naopak nezdařila (0).

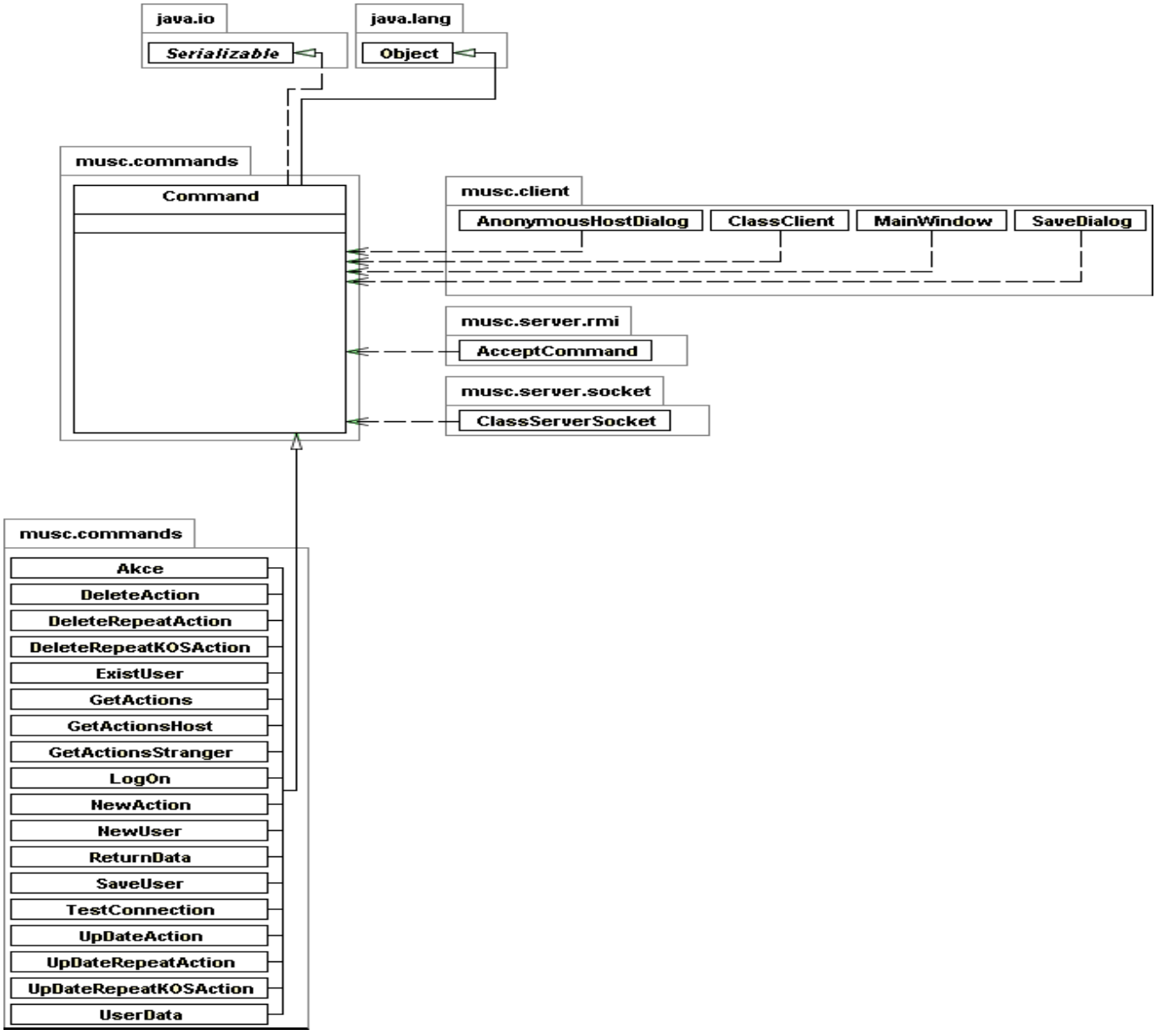
**Třída ClassDatabase**

Třída si uchovává v instančních proměnných informace o typu databáze, adrese a portu databáze, názvu databáze, uživatelském jménu a heslu do databáze. Metoda createConnectionToDatabase() slouží k navázání spojení s příslušnou databází. Na druhé straně metoda close() slouží k ukončení spojení s databází. Třída poskytuje celou řadu metod, které slouží pro specifické požadavky na databázi. Protože těchto požadavků je celkem 16, uvedu zde jenom některé. Např. metoda newUser(NewUser news,ReturnData ret) slouží k vytváření nových uživatelských účtů. Parametr news obsahuje všechny potřebné údaje pro registraci uživatele, jako je uživatelské jméno, heslo atd. Parametr ret slouží k uložení návratového kódu celé operace do jeho instanční proměnné state a pro uložení dat získaných z databáze do jeho instanční proměnné array. Nejprve je provedena kontrola zda uživatelem zvolené uživatelské jméno je unikátní. Pokud ne, je celá operace ukončena a jako výsledek operace je vrácena hodnota 2, které indikuje již existující uživatelské jméno. V kladném případě je uživateli přiděleno jeho unikátní identifikační číslo. Následně dojde k zašifrování hesla a poté je provedena registrace uživatele.

Další ukázkou je pak metoda newAction(NewAction s,ReturnData ret), která slouží k uložení nových akcí do databáze. Nejprve jsou připraveny všechny dotazy do databáze s využitím instance třídy java.sql.PreparedStatement. Následně jsou všechny dotazy do databáze odeslány v rámci jedné dávky. Díky tomu, že jsou dotazy odesílány po dávkách, dochází k zvýšení efektivnosti v komunikaci s databází a jejímu následnému nižšímu zatěžování.

## 6.2 Komunikace mezi klientem a serverem

Komunikace mezi klientem a serverem je zajištěna pomocí klasických socketů. Byla zvolena komunikace pomocí dávek příkazů. Komunikace mezi serverem a klientem tedy probíhá pomocí příkazů. Tyto příkazy jsou v programu reprezentovány skrze objekty. Každá třída, jejíž instance tvoří objekt zajišťující komunikaci, dědí od třídy Command. Třída Command implementuje rozhraní Serializable. Rozhraní Serializable zajišťuje serializaci objektů na straně odesilatele a následnou deserializaci objektů na straně příjemce. Diagram třídy Command lze vidět na obrázku 8.




**Obrázek 8: UML diagram – Diagram tříd třídy Command**

Typů příkazů je celkem 18 druhů. Např. třída LogOn reprezentuje příkaz přihlášení do systému. Nebo třída GetActions představuje příkaz pro získání dat, v tomto případě plánu reprezentovaného jednotlivými akcemi, z databáze. Další třídou je třeba NewUser, která slouží jako příkaz pro vytvoření nového uživatelského účtu. Speciálním typem je třída ReturnData.

Instance této třídy slouží k zasílání výsledků požadovaných operací klientovi. Tato třída má dvě instanční proměnné. První proměnná je typu integer a jmenuje se state. Proměnná state slouží k uložení návratového kódu operace. Druhou proměnnou je array typu java.util.ArrayList. Tato proměnná slouží k ukládání a přenosu dat získaných během operace.

Na straně klienta je v okamžiku, kdy je požadována nějaká služba od serveru, vytvořen objekt od konkrétní třídy, která dědí od třídy Command, reprezentující příkaz. Protože obvykle dochází k žádosti o více služeb najednou ze strany klienta, jsou tyto příkazy nejprve ukládány do dočasného objektu typu java.util.ArrayList. Pokud jsou shromážděny všechny požadavky, dojde k navázání komunikace mezi klientem a serverem. V tento okamžik jsou odesílané objekty zašifrovány pomocí algoritmu Blowfish. Více o zabezpečení komunikace v kapitole 6.8 Bezpečnost programu MUSC. Následně je navázáno spojení a takto šifrované objekty jsou odeslány.

Na straně serveru jsou tyto objekty nejprve dešifrovány. Dešifrací získáme objekty typu Command. Následně je provedena dynamická identifikace objektu a určen konkrétní typ objektu. Na základě takto identifikovaného objektu je následně provedena požadovaná akce. Ve většině případů se jedná o požadavek vyžadující nějakou manipulaci s daty v databázi. V tento okamžik je navázáno spojení s databází. Poté je provedena interakce s databází. K navrácení výsledku požadované operace je použita instance třídy ReturnData. Návratový kód požadované operace je uložen do proměnné state a požadovaná data do proměnné array obsažené v třídě ReturnData. Po vyřízení všech požadovaných služeb je instance typu ReturnData, která obsahuje návratový kód operace a požadovaná data, zašifrována a odeslána klientovi nazpět.

Klient načte příchozí data a dešifruje je. Provede se dynamická kontrola zda jde o objekt typu ReturnData. Poté je zkontrolován návratový kód operace. Pokud operace proběhla úspěšně, jsou z proměnné array získána data, která jsou výsledkem požadované operace.

## 6.3 Komunikace s databází

Při spuštění serverové aplikace je zaregistrován ovladač pro databázi k níž se bude server připojovat. V případě komunikace s MySQL databázovým serverem bude registrace vypadat následovně:

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Pokud je na server zaslán požadavek, který vyžaduje komunikaci s databází, je vytvořena instance třídy ClassDatabase. V konstruktoru ClassDatabase (String dat,String type,String u,String pass,String hostDat,String portDat,int max,Cipher encrypt,Cipher decrypt) je předáno jméno databáze, typ databázového serveru, uživatelské jméno a heslo pro připojení k databázi, síťová adresa na které je databázový server a port na kterém poslouchá. Dále je v konstruktoru předán maximální počet současně vytvořených akcí a objekty šifry pro šifrování a dešifrování hesla uživatele v databázi. Parametry předané v konstruktoru jsou uloženy do instančních proměnných třídy ClassDatabase. Pro komunikaci s databází jsem použil třídy z balíčku java.sql.

Následně je zavolána metoda createConnectionToDatabase(), která naváže spojení s databází. Tato metoda vrátí jako návratový kód informaci zda se podařilo navázat spojení s databází. Pokud se připojení zdaří, je uložena reference na toto spojení do proměnné connection, která je datového typu java.sql.Connection. V případě MySQL databáze bude navázání spojení vypadat následovně:

```
connection = DriverManager.getConnection("jdbc:mysql://" + hostDatabase +":" +
                portDatabase + "/" + database + "?user=" + user +"&password=" + password);
```

Obvykle požadavek na databázi obsahuje více dotazů než jen jeden. Z tohoto důvodu by nebylo efektivní odesílat každý dotaz zvláště. Výhodné by určitě bylo předpřipravit si všechny požadované dotazy a následně je odeslat v rámci jedné dávky do databáze. Rozhraní java.sql.Connection poskytuje metodu prepareStatement(String sql). Tato metoda jako parametr přijímá SQL dotaz a vrací referenci na objekt PreparedStatement. Rozhraní PreparedStatement

dědí od rozhraní Statement. V případě vykonání dotazu pomocí rozhraní Statement dochází ke kompilaci odesílaného SQL dotazu JDBC ovladačem při každém odeslání. Naopak při použití metody metody pro vykonání dotazu z rozhraní PreparedStatement je daný SQL dotaz kompilován pouze jednou.

Další nespornou výhodou rozhraní PreparedStatement je, že při vytvoření SQL dotazu je vložena pouze kostra SQL dotazu. Tato kostra SQL příkazu pak může být doplněna pomocí substitučních parametrů.

Poslední velkou výhodou je, že o zpracování SQL dotazu se nestará programátor, ale ovladač JDBC. Díky této skutečnosti lze vložit do parametrizovaného SQL dotazu i řetězce obsahující neobvyklé znaky, jako je např. apostrof, které by při klasickém vytváření dotazů pomocí rozhraní Statement mohly způsobit chybu.

Jak již bylo naznačeno, je vhodné, aby velké množství dotazů do databáze bylo odesláno najednou. Zde jsem využil metodu z rozhraní PreparedStatement addBatch(). Po vykonání této metody je již předpřipravený příkaz uložen do vyrovnávací paměti. Poté co jsou takto zpracovány všechny předpřipravené příkazy, je zavolána metoda executeBatch() z rozhraní PreparedStatement. Po jejím zavolání jsou provedeny všechny dotazy do databáze. Jako návratový kód vrací pole s počtem změněných řádků v databázi. Tento způsob komunikace je tedy vhodný především pro vytváření, změnu a mazání záznamů v databázi.

Při požadavku na vytvoření nového uživatelského účtu je potřeba provést několik po sobě následujících dotazů do databáze. Nejprve je třeba zjistit zda uživatelem požadované uživatelské jméno doposud neexistuje. Pokud dané uživatelské jméno doposud neexistuje je třeba zjistit nové unikátní identifikační číslo pro daného uživatele. Nakonec je potřeba zanést do databáze všechny údaje o novém uživateli. Tyto všechny akce je nutné provést dohromady jako jednu atomickou operaci. V případě selhání některé dílčí operace, je nutné vrátit databázi do původního stavu před započetím vykonávání první operace. Tento a jemu podobné požadavky na databázi jsem proto řešil transakčním zpracováním.

V Jave dochází k potvrzení po každém vykonání dotazu do databáze. Zde je však třeba, aby byla potvrzena až celá skupina příkazů do databáze. V této situaci lze zavolat metodu setAutoCommit(boolean autoCommit) z rozhraní java.sql.Connection. Při zavolání této metody s parametrem false, dojde k zrušení potvrzování po každém vykonaném dotazu do databáze. Nyní lze provést posloupnost dotazů, které musí být provedeny jako jeden celek. Po vykonání všech dotazů je třeba zavolat metodu commit() z rozhraní java.sql.Connection. Po zavolání této metody dojde k potvrzení všech dříve provedených dotazů do databáze, které nebyly doposud potvrzeny. Pokud nastane při zpracování a následném potvrzení dotazů chyba, je třeba zavolat metodu rollback() z rozhraní java.sql.Connection. Po zavolání této metody je databáze uvedena do původního stavu před započetím provádění příkazů z aktuální transakce.

## 6.4 Načítání a sdílení plánů ostatních uživatelů

Aplikace umožňuje prohlížení plánů ostatních uživatelů. Tato funkce je dostupná skrze položku „Nahrát cizí plán“ v menu Uživatel. Po zvolení této nabídky je třeba zadat uživatelské jméno požadovaného uživatele. Po jeho zadání je vytvořen objekt třídy ExistUser, která dědí od třídy Command. Objekt této třídy slouží k ověření existence požadovaného uživatele. Jako parametr do konstruktoru je předáno uživatelské jméno požadovaného uživatele. Poté je zavolána statické metoda createConnection() třídy ClassClient, která naváže spojení se serverem. Po navázání spojení je na server odeslán objekt ExistUser. Na straně serveru je příchozí objekt identifikován jako objekt třídy ExistUser. Protože seznam uživatelů se nachází pouze v databázi je potřeba vytvořit objekt třídy ClassDatabase. Následně je zavolána statická metoda createConnectionToDatabase() třídy ClassDatabase, která naváže spojení s databází. Poté je proveden dotaz do databáze, zda daný uživatel existuje.

```
SELECT UZIVATEL.*
FROM UZIVATEL
WHERE UZIVATEL.UZIVATELSKE_JMENO =  ExistUser.user
```

Pokud daný uživatel existuje, jsou z databáze získány jeho osobní údaje a uloženy do objektu UserData. Následně je nastaven návratový kód úspěšného vykonání operace na 1 a jsou uložena všechna data získaná při zpracování příkazu ExistUser do objektu datového typu ReturnData. Jestliže však uživatel neexistuje je návratový kód operace nastaven na 2. Následně je klientovi odeslán výsledek operace.

Klient příjme odpověď ze serveru a zkontroluje návratový kód. Pokud se rovná 2, tak požadovaný uživatel neexistuje a uživateli je tato skutečnost sdělena. Pokud je návratový kód 1, je zjištěn typ aktuálního zobrazeného plánu. Aktuální zobrazený plán je zjištěn, jelikož je třeba určit pro jaké období má být plán cizího uživatele získán. Každý plán může nezávisle na ostatních plánech zobrazovat jiné časové období, a tak je třeba určit konkrétní časové období. Poté co je určeno časové období, na které má být plán nahrán, je vytvořen objekt třídy GetActionsStranger, který slouží k získání plánu cizího uživatele. V konstruktoru je předáno aktuální datum aktivního plánu (rok, měsíc a den), dále pak identifikační číslo požadovaného uživatele a přihlášeného uživatele. Aby se zamezilo příliš častým přístupům do databáze a posílání dotazů na server, je vždy načten časový plán na tři měsíce (aktuální, předchozí a následující měsíc). Z tohoto důvodu dojde v rámci konstruktoru objektu GetActionsStranger, k vypočtení předchozího a následujícího měsíce od data předaného jako parametr konstruktoru.

Poté je navázáno spojení se serverem a objekt s požadavkem na získání akcí od cizího uživatele je odeslán. Na serveru je objekt přijat a je identifikován typ požadavku. Následně je vytvořen objekt třídy ClassDatabase a je navázáno spojení s databází. Nejprve je vykonán dotaz na získání seznamu skupin, kterých je členem přihlášený uživatel.

```
SELECT UZIVATEL_SKUPINA.SKUPINA
FROM UZIVATEL_SKUPINA
WHERE UZIVATEL_SKUPINA.R8_ID_UZIVATEL = GetActionsStranger.id_userLogon
```

Poté je vykonán dotaz na získání seznamu skupin, do kterých patří požadovaný uživatel. Nakonec je proveden dotaz do databáze na získání akcí, které patří cizímu uživateli a mají počáteční rok a měsíc shodný s požadovaným časovým obdobím, na které má být plán cizího uživatele nahrán. Dotaz vybere pouze ty akce, které náleží do skupin jejichž členem je jak přihlášený uživatel tak i cizí uživatel, kromě skupiny PRIVATE. Tento dotaz je proveden pro všechny tři požadované měsíce (aktuální, předchozí a následující). Takto získaná data jsou pak uložena do objektu třídy ReturnData. Pak je nastaven návratový kód operace a výsledek operace je odeslán klientovi zpět.

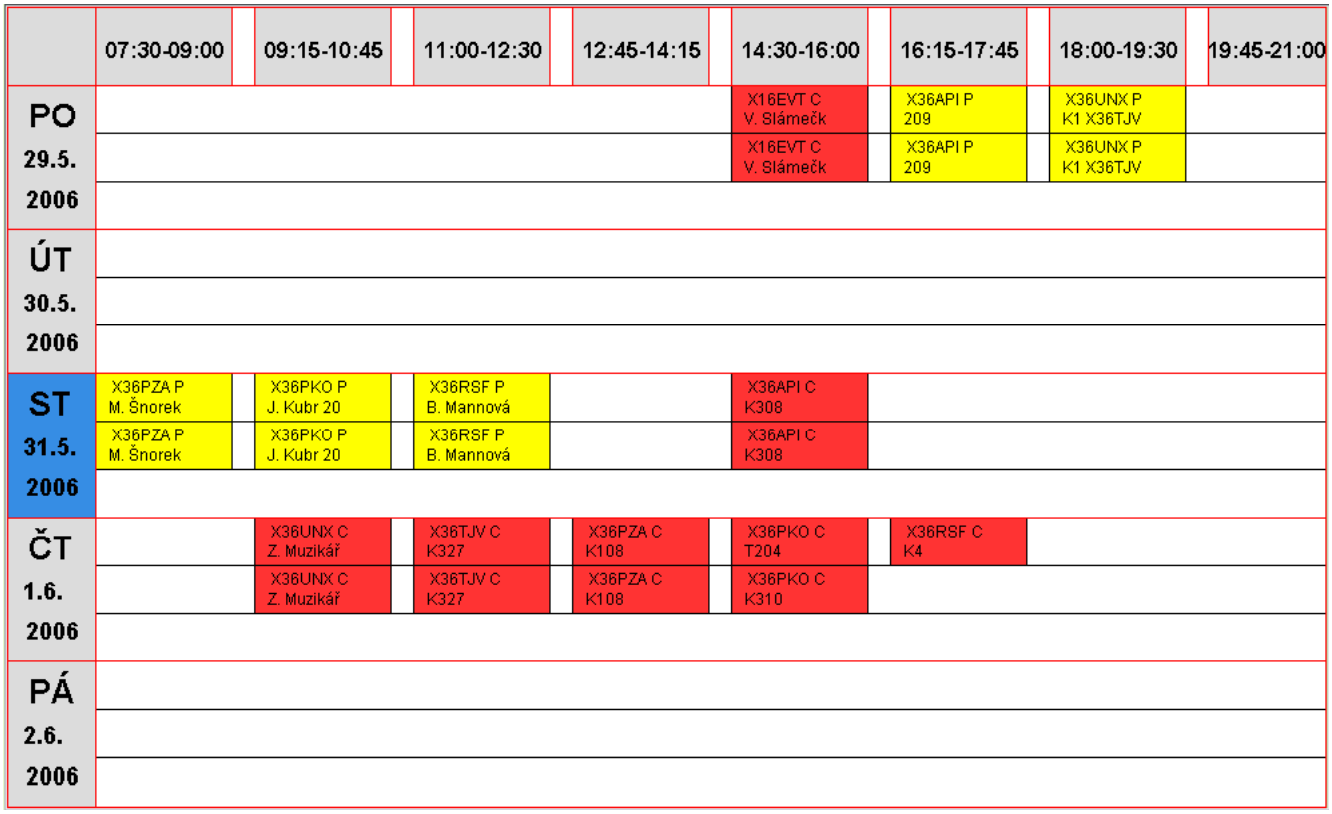
Klient přijme výsledek operace a zkontroluje návratový kód. Pokud je roven 1 dopadla operace úspěšně. Jsou vybrána získaná data a jsou vložena do vyrovnávací paměti buffer, která je statická proměnná třídy DataBuffer. Následně je buffer seřazen pomocí statické metody seradDataBuffer() třídy DataBuffer. Nakonec je získaný plán cizího uživatele vykreslen.

Akce, které náleží cizímu uživateli, nemůže přihlášený uživatel měnit nebo mazat. O cizích akcích mohou být zobrazeny veškeré informace. Pokud jsou při identifikaci akcí nalezeny akce, které náleží cizímu uživateli, není pro ně vyvolán dialog pro editaci a mazání akcí. Aplikace umožňuje pro jedno časové období nahrát a zobrazit neomezeně cizích uživatelů. Při přechodu na jiné časové období je nahrán pouze plán přihlášeného uživatele a posledního nahrávaného cizího uživatele.

## 6.5 Vykreslování dat

### 6.5.1 Vykreslování dat v jednotlivých pohledech

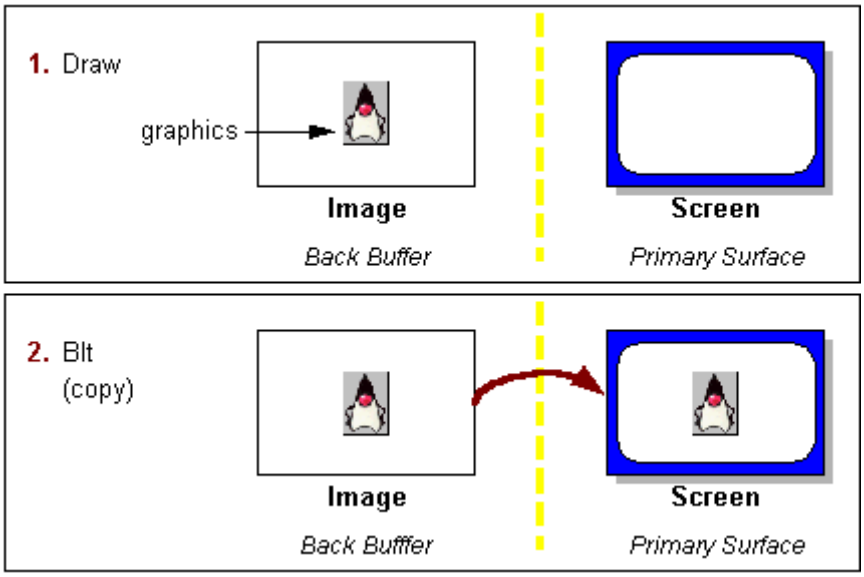
Pro vykreslení dat, v tomto případě jednotlivých akcí, je ve všech pohledech (Denní plán, Detailní denní plán, KOS plán, Týdenní plán a Měsíční plán) použit stejný algoritmus. Jednotlivé akce jsou vykreslovány podle jejich uspořádání, které je dané jejich počátečním datem a časem. Data jsou umístěny v dočasném bufferu, který je typu java.util.ArrayList. Data v tomto bufferu jsou seřazena podle počátečního data a času jednotlivých akcí. Buffer je řazen pouze v okamžiku kdy dojde k přidání nových dat do bufferu nebo v případě jejich modifikace. V případě odebrání není třeba provádět nové seřazení dat, protože ArrayList je lineární struktura a po odebrání prvku dojde k posunutí všech následujících prvků. Při vykreslování jsou jednotlivé akce postupně vybírány z tohoto bufferu a použity k vykreslení. Vždy je vykreslován každý den zvlášť. Poté co je akce vyjmuta z bufferu je z kontrolováno zda akce trvá pouze v rámci právě vykreslovaného dne. Pokud ano, je tato akce vykreslena. Pokud však akce trvá více než jeden den, je třeba určit, zda část akce, jenž se odehrává ve vykreslovaném dni, v tento den začíná, končí nebo se vyskytuje uprostřed období, po které akce trvá. V případě, že akce ve vykreslovaný den začíná, je tato akce vykreslena od časového okamžiku, ve který začíná, do nejzažšího času pro daný pohled. Pokud akce v daný den končí, je akce vykreslena od prvního zobrazitelného času pro daný plán až do koncového času akce. Jestliže akce je ve střední části celého období trvání akce, je tato akce vykreslena od prvního zobrazitelného času do nejzažšího časového okamžiku pro konkrétní plán.




**Obrázek 9: KOS plán**

Ve všech pohledech existuje možnost zobrazení více současně probíhajících akcí (počet se liší podle jednotlivých plánů). V tomto případě bylo potřeba zajistit dvě věci. První věcí je, aby se souběžné akce při vykreslování nepřekrývali. Druhou pak je, aby v případě že se vyskytly dvě souběžné akce, byly další vykresleny tak, že následují za první vykreslenou akcí z dvojice souběžných akcí. Pokud by nebylo toto zajištěno, tak v případě výskytu souběžných akcí by celý zbytek vykreslovaných událostí klesl o úroveň níže v rámci zobrazení. Z tohoto důvodu bylo třeba jednotlivé akce vykreslovat podle jejich časového pořadí.

Každý den je rozdělen do několika souběžných časových linií. Počet časových linií určuje maximální počet souběžně vykreslovaných akcí pro daný pohled. Tento počet se liší u jednotlivých pohledů, např. u Detailního denního plánu je to sto, ale např. u Týdenního plánu jsou to tři. V rámci každé časové linie je značka, která určuje koncový časový okamžik poslední vykreslené akce. V okamžik kdy má být akce vykreslena, se začnou procházet jednotlivé časové linie, v pořadí od první do poslední. Jakmile je nalezena první časová linie, v níž je počáteční čas akce, která má být vykreslena, roven nebo větší časové značce pro danou linii, je tato událost vykreslena. Ostatní linie se už dále neprochází. Pokud by se vyskytla situace, že pro akci nelze nalézt žádnou časovou linii, tak se tato akce nevykresluje. Tato situace by mohla např. nastat pokud máme čtyři souběžné akce a např. Týdenní plán umožňuje zobrazit maximálně tři současně probíhající akce. Díky skutečnosti, že akce jsou seřazeny podle počátečního data a času, dochází k optimalizaci časového plánu. Plán je optimalizován tak, že jsou v maximální možné míře využita prázdná místa mezi akcemi v plánu, kde se nic neděje, pro zaplnění ostatními souběžně začínajícími nebo probíhajícími akcemi. Jak vypadá vykreslený pohled, lze vidět na obrázku 9.




**Obrázek 10: Double-buffering**

Tento způsob vykreslování je jednoduchý a zcela splňuje kladené požadavky. Nejjednodušším způsobem jak vykreslit námi vytvořený obrázek je např. po pixelech, úsečkách, kružnicích apod. Problémem však je, že je to velmi časově náročné a neefektivní. Tato skutečnost způsobuje, že vykreslování plánu trvá dlouho a lze tak proto vidět artefakty jak se obrázek postupně vykresluje. Dále dochází také k blikání obrazu. Navíc k výše popisovanému postupu dochází při každém vykreslování obrazovky, ať již vyvolaném úmyslně programem nebo vyvolaném ve spolupráci operačního systému s aplikací, např. v případě posunutí okna s aplikací. Z tohoto důvodu jsem použil techniku zvanou double-buffering. V tomto případě se vytvoří pomocí grafického objektu tzv. offscreen obrázek, který se vytvoří v paměti. Následně ve stejném kroku zavolám metodu drawImage() a vykreslím obrázek vytvořený v paměti na obrazovku. Plocha obrazovky se obvykle vztahuje k primární vrstvě a offscreen obrázek, který je použitý pro double-

buffering, se vztahuje k zadnímu bufferu. Situace kdy se kopíruje obsah jedné vrstvy na druhou se označuje jako blokový řádkový přenos nebo blitting („blt"). Aby však bylo dosaženo ještě vyšší výkonnosti, je plán na tento obrázek vykreslován pouze v případě, kdy je to vyžadováno programem. Tato situace nastává např. při tvorbě, editaci nebo smazání akce. V ostatních případech, kdy není potřeba měnit zobrazovaný plán, je vykreslen pouze starý obrázek. Tato situace nastává např. když uživatel tahem myši vytváří novou akci. Plán je vykreslen ze starého obrázku a nově je vykreslen pouze obdélník určující vytvářenou akci. Technika double-bufferingu je znázorněna na obrázku 10.

### 6.5.2 Detekce a identifikace akcí v jednotlivých pohledech

Při práci s časovým plánem je potřeba jednotlivé akce vytvářet, editovat a mazat. Aby bylo možné nějakou konkrétní akci editovat, je třeba ji nejprve identifikovat. Uživatel akci identifikuje pomocí kliknutí myši a tedy jediný údaj, který lze získat z této události, jsou souřadnice místa, kam uživatel klikl. Samotné objekty reprezentující akce jsou však pouze vykresleny a dále již nemají žádnou spojitost s plochou na níž jsou vykresleny. Jinými slovy řečeno, zobrazovací plocha neví o žádných akcích, které jsou na ni vykresleny.

Navrhl jsem tedy třídu SouradniceAkce, jejíž objekt uchovává objekt třídy Akce a souřadnice dané akce při jejím vykreslení. Uchovávají se souřadnice levého horního rohu a souřadnice pravého dolního rohu. Během vykreslování akcí na offscreen obrázek do paměti, se pro akce, které byly vykresleny, vytvoří objekt typu SouradniceAkce. Následně je tento objekt vložen do proměnné array typu java.util.ArrayList. Proměnná array slouží k uchování objektů typu SouradniceAkce pro všechny vykreslené akce.

Objekt typu javax.swing.JPanel, na který se vykreslují jednotlivé plány, si zaregistruje objekt pro zachytávání událostí od myši. Tento objekt implementuje rozhraní MouseListener. Z tohoto rozhraní je pak třeba přepsat metodu mousePressed(MouseEvent e). Pokaždé když uživatel zmáčkne tlačítko myši nad objektem, na který se vykresluje plán, je vyvolána metoda mousePressed(MouseEvent e). Objekt typu MouseEvent nese řadu informací o vzniklé události. Jednou z informací jsou i souřadnice kurzoru myši v okamžik zmáčknutí tlačítka. Takto získané souřadnice myši jsou použity pro případnou identifikaci akce. Tyto souřadnice jsou postupně porovnávány se souřadnicemi uloženými v objektech typu SouradniceAkce, které byly vytvořeny při vykreslení plánu. Pokud není nalezena žádná akce, pro kterou by souřadnice získané z události zmáčknutí tlačítka myši odpovídaly souřadnicím akce, tak je celý proces hledání ukončen a nic se dál neděje. V opačném případě, kdy je akce nalezena, tak se tato akce vyzvedne z proměnné SouradniceAkce a je otevřen dialog pro editaci a mazání akcí.

### 6.5.2 Nastavení aplikace

Aplikace MUSC umožňuje uživateli, aby si nastavil barevné schéma aplikace dle svých požadavků. Toto barevné schéma umožňuje nastavit jak vzhled samotné aplikace, tak i vlastnosti pro zobrazování dat. Nastavení je uloženo v externím souboru MUSCoptions.xml, více se o jeho struktuře dozvíte v kapitole 6.7.2 Popis datového formátu pro uložení nastavení aplikace. Toto nastavení je načteno vždy při startu aplikace. Uživatel má dvě možnosti jak změnit nastavení aplikace. Prvním způsobem je přímá editace souboru MUSCoptions.xml, což však není uživatelsky příliš přívětivé. Druhou variantou je změna nastavení přímo v aplikaci.

V menu Nastavení lze nalézt celkem čtyři položky – Barevné nastavení aplikace, Barevné nastavení priorit, Původní nastavení a Uložit nastavení. Položka Barevné nastavení aplikace umožňuje nastavit barvu pozadí v jednotlivých pohledech, změnit barvu záhlaví v jednotlivých pohledech nebo barvu pro označení aktuálního dne. Dále lze také nastavit podklad v Měsíčním plánu pro zobrazení informace výskytu jednodenních akcí. Také je možné změnit barvu pro zvýraznění dne při přechodu v Měsíčním plánu, po kliknutí pravého tlačítka myši, do Týdenního plánu.

Položka Barevné nastavení priorit umožňuje nastavit barvu pro zobrazení akce s danou

prioritou. Další položkou je Původní nastavení, které slouží k nastavení původního barevného schéma aplikace. Poslední položkou je pak Uložit nastavení, které uloží právě aktuální nastavení.

Při inicializaci aplikace je vytvořen objekt třídy ColorOptions. Následně je volána statická metoda loadUserOptins(), která načte soubor s nastavením aplikace. Nastavení z tohoto souboru je uloženo do statických proměnných třídy ColorOptions. Třída ColorOptions obsahuje statické proměnné, které uchovávají aktuální nastavení aplikace. Jedná se o tyto proměnné – headerColor, backgroundColor, selectedDayColor, todayColor a oneDayMonthColor. Všechny tyto proměnné jsou datového typu java.awt.Color. Dále třída ColorOptions obsahuje privátní statické pole priorities [ ], které je datového typu java.awt.Color. V tomto poli jsou uloženy barvy pro zobrazení jednotlivých priorit. Aby bylo možné získat hodnoty barev pro jednotlivé priority, je zde statická metoda getColorPriority(int i). Tato metoda jako parametr přijímá číslo priority a navrací barvu pro danou prioritu (objekt typu java.awt.Color). Třída také poskytuje statickou metodu pro změnu barvy priority setColorPriority(int i,Color c), jejíž parametry určují číslo priority a novou barvu pro danou prioritu.

Třída ColorOptions ještě poskytuje metodu setDefaultColorsPriority() a setDefaultColors(). Metoda setDefaultColorsPriority() slouží k nastavení původních barev pro jednotlivé priority. Metoda setDefaultColors() pak umožňuje nastavit počáteční barevné nastavení samotné aplikace.

Nastavení aplikace je pak využíváno při vykreslování jednotlivých pohledů. Protože o vykreslování jednotlivých plánů se starají nezávislé třídy, je třeba zajistit dostatečně efektivní přístup k tomuto nastavení. Vytvářet pro každou třídu starající se o vykreslování plánů samostatný objekt třídy ColorOptions by bylo neefektivní. Z tohoto důvodu je nastavení uložené ve třídě ColorOptions získáno skrze statické proměnné a statické metody. Obdobným způsobem je řešena i změna nastavení.

## 6.6 Funkce Import rozvrhu z KOSu

Jednou ze služeb, kterou poskytuje program MUSC, je import rozvrhu z KOSu. Tento rozvrh však musí být v HTML formátu generovaném samotným KOSem. Pro provedení importu rozvrhu musí uživatel v menu zvolit položku „Importovat rozvrh z KOSu“. Následně musí uživatel udat polohu souboru s rozvrhem na lokálním disku klientské aplikace. Poté musí uživatel zadat časové období na které má být rozvrh importován. Pak je provedena kontrola vstupního souboru s rozvrhem. Pokud je soubor v pořádku, tak je proveden import rozvrhu z KOSu. Tento plán je následně uložen do lokální vyrovnávací paměti. Pro trvalé uložení rozvrhu je třeba provést volbu „Uložit plán“, jinak by po ukončení programu MUSC byl plán natrvalo ztracen a import by musel být proveden znovu.

KOS (Komponenta Studia) generuje pro každého zaměstnance (studenta, učitele atd.) rozvrh v podobě webové stránky v HTML kódu. Protože nelze získat přímý přístup do databáze KOSu ani nelze žádným jiným způsobem získat rozvrhy všech členů akademické obce (zaměstnanci, učitelé, studenti atd.) ČVUT hromadně, bylo rozhodnuto, že si import rozvrhu z KOSu musí každý uživatel provést sám. Adresa na níž KOS umisťuje rozvrhy však bohužel není generována na základě nějakých zřejmých pravidel (např. podle uživatelského jména, předmětu atd.), a tak nelze ani provést variantu, kdy by uživatel zadal údaje potřebné pro připojení do KOSu a MUSC by rozvrh stáhl sám. Proto bylo přistoupeno k variantě, že si uživatel rozvrh uloží sám na lokální disk a provede import rozvrhu do aplikace.

### 6.6.1 HTML parser

Cílem je vytvoření HTML parseru pro analýzu zdrojového kódu v HTML. Tato analýza je využita při importu rozvrhu z KOSu, kdy uživatel poskytne k analýze soubor v HTML formátu se svým rozvrhem. Z celého souboru jazyka HTML verze 4.1 je vybrána pouze podmnožina tagů – BR | HR | HTML | BODY | HEAD | TITLE | TABLE | TR | TD | TH | A | SMALL | P | /HTML | /BODY | /HEAD | /TITLE | /TABLE | /TR | /TD | /TH | /A | /SMALL | /P a parametrů – NOSHADE | SIZE | WIDTH | ALIGN | SUMMARY | BORDER | BGCOLOR | CELLPADDING | CELLSPACING | VALIGN | ABBR | ROWSPAN | COLSPAN | HEIGHT | HREF | NAME. Tato skupina tagů a parametrů byla vybrána z důvodu, jejich výskytu ve zdrojový kódech, které generuje KOS při tvorbě rozvrhů. Program nejprve provede rozdělení zdrojového kódu na jednotlivé lexikální elementy. Následně program získá potřebná data pro import rozvrhu do systému, jako jsou údaje o události, o času (od kdy do kdy je vyučování), číslo místnost kde se vyučuje atd.

### 6.6.2 EBNF

Pro vytvoření HTML parseru bylo potřeba vytvořit EBNF (rozšířená Backus Naurova forma). Tato EBNF sloužila pro vytvoření přehledu o syntaktické struktuře HTML dokumentu. Na základě takto vytvořené EBNF byla určena syntaxe jednotlivých lexikálních elementů.

**Pozn.** Malými písmeny jsou označeny terminály, velkými písmeny jsou označeny neterminály. Jednotlivé lexikální elementy jsou odděleny mezerou.

R → < html > {E | K} P {E | K} [Q] </html > eof
Q → e | < body [bgcolor] > {L | E | K} </body >
E → text
K → kom
P → < head > {E | K} O {E | K} </head >
O → < title > {E | K} </title >
L → < M | A | B | C | D | F | I | J >
M → br | hr {A1}
A → table {A2} > {L | E | K} </table

B → tr {A3} > {L | E | K} </tr
C → td {A4} > {L | E | K} </td
D → th {A4} > {L | E | K} </th
F → a {A5} > {L | E | K} </a
I → small > {L | E | K} </small
J → p > {L | E | K} </p
A1 → noshade | size | width | align
A2 → summary | align | width | border | bgcolor | cellpadding | cellspacing
A3 → bgcolor | valign
A4 → abbr | rowspan | colspan | width | align | height | valign | bgcolor
A5 → href | name

### 6.6.3 Lexikální elementy

Za pomoci EBNF byla stanovena syntaxe základních lexikálních elementů. Mezi základní lexikální elementy patří: TEXT a SPECIÁLNÍ SYMBOL. Lexikální element TEXT lze složit z elementů písmeno, číslice a znak. Lexikální element SPECIÁLNÍ SYMBOL je tvořen lexikálním elementem TAG, PARAMETR, LPAR, RPAR, EOF nebo KOM.

#### 6.6.3.1 Syntaxe lexikálních elementů

lexikální element = TEXT | SPECIÁLNÍ SYMBOL

TEXT = písmeno{písmeno | číslice | znak}

písmeno = A | B | ... | Z | Ž | a | b | ... | z | ž

číslice = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

znak = | ` | ~ | & | @ | _ | [ | ] | { | } | ( | $ | ^ | * | + | | | \ | \ | | ? | . | , | °| § | ) | ! | - | : | / | . | = |
" | < | > | % | # | ; |

SPECIÁLNÍ SYMBOL = TAG | ATRIBUT | LPAR | RPAR | EOF | KOM

TAG = br | hr | html | body | head | title | table | tr | td | th | a | small | p | </html | </body |
</head | </title | </table | </tr | </td | </th | </a | </small | </p

PARAMETR = noshade | size | width | align | summary | border | bgcolor | cellpadding |
cellspacing | valign | abbr | rowspan | colspan | height | href | name

LPAR = <

RPAR = >

KOM = LPAR!--TEXT--RPAR

Jednotlivé lexikální elementy jsou získávány třídou SyntaktickyAnalyzator, která vždy požádá instanci třídy LexikalniAnalyzator o další lexikální element ze vstupu, který je v tomto případě představován HTML souborem. Lexikální analyzátor je implementován jako konečný automat, který slouží k rozpoznávání jednotlivých lexikálních elementů ze vstupních dat.

### 6.6.4 Získání dat z HTML Parseru

O získání dat se stará třída SyntaktickyAnalyzator. Data jsou získána na základě klíčových slov vyskytujících se v textu (po, pá, st, čt, pá, pondělí, úterý, středa, čtvrtek, pátek, sobota, neděle, mo, tu, th ,we, th, fr), která oddělují jednotlivé dny v rozvrhu. Dále se v textu vyskytují dvě klíčová slova – „lichý“ a „sudý“. Tyto dvě klíčová slova označují zda jde o rozvrh na lichý nebo sudý týden. Tato klíčová slova jsou použita na rozlišení rozvrhu v jednotlivých týdnech. Jednotlivé údaje (předměty) jsou reprezentovány samostatnými buňkami v tabulce. Dalším klíčovou skutečností pro získání dat je výskyt parametru „COLSPAN“, který udává počet sloupců, které zabírá jedna buňka (předmět). Jednomu dni odpovídá patnáct sloupců, takže na tradiční dobu výuky, tj. 1.5 hodiny připadají dva sloupce. Z textu vyskytujícím se v HTML kódu jsou vybrány texty, které se vyskytují po klíčových slovech označujících den, klíčovém slovu „COLSPAN“ a klíčových slovech označujících lichý a sudý týden. Tyto údaje jsou ukládaná do datové struktury array, která je typu java.util.ArrayList. Poté jsou získaná data zpracována a upravena do stejné formy jako jsou data v programu MUSC.

#### 6.6.4.1 Import rozvrhu z KOSu

KOS generuje pro každého zaměstnance rozvrh v podobě webové stránky v HTML kódu. Uživatel si může tento rozvrh uložit jako webovou stránku. MUSC podporuje funkci importu rozvrhu z KOSu v podobě HTML stránky, která je generována samotným KOSem. Uživatel určí místo uložení rozvrhu na disku svého počítače. Následně je vyzván, aby zadal počáteční datum a koncové datum období, na které má být rozvrh z KOSu importován. Počáteční datum určuje začátek semestru a koncové datum určuje konec semestru. Uživatel ale není nijak omezován ve zvoleném období na nějž má být rozvrh importován. Z čehož plyne, že uživatel si může importovat rozvrh třeba na jeden měsíc, týden či den nebo na libovolné období. Rozvrh trvá vždy pravidelně celý pracovní týden od pondělí do pátku. Díky tomu, že lze zadat přesné datum počátku a konce semestru, bere importér v potaz existující problém, a to že semestr většinou nezačíná v pondělí a nekončí v pátek, takže se importovaný rozvrh přizpůsobí požadovanému časovému období a importuje rozvrh, který je přesně ohraničen daty zadanými uživatelem.

Nejprve je zjištěno zda doba na níž má být rozvrh importován je v rámci jednoho týdne. Pokud je období importu v rámci jediného týdne je určen počáteční a koncový den v rámci týdne (po, pá, st, čt, pá, pondělí, úterý, středa, čtvrtek, pátek, sobota, neděle, mo, tu, th ,we, th, fr). Následně jsou z dat uložených v datové struktuře java.util.ArrayList vybrána pouze data v období mezi počátečním a koncovým datem. Následně je provedeno vytvoření dat pro systém viz. Kapitola 6.6.4.2 Vytvoření dat pro Víceuživatelský plánovací kalendář.

Pokud je rozvrh importován na období delší než jeden týden (v celku), je vygenerován rozvrh v prvním týdnu, který může být celý, ale ve většině případů je zkrácený. Proto jsou nejprve z dat získaných za pomoci analyzátoru a uložených v datové struktuře java.util.ArrayList, vybrána pouze data vyskytující se v období mezi počátečním dnem a koncovým dnem týdne (neděle). Z datové struktury obsahující data získaná za pomoci analyzátoru jsou vybrána pouze data, která jsou potřeba pro generovaní prvního týdne, obvykle jen jeho části. Poté se provede generovaní akcí pomocí třídy KOSDataAkce, která vygeneruje data ve formátu požadovaném systémem MUSC.

Následně je provedeno vytvoření akcí v rozvrhu ve všech následujících týdnech. Opět je využita třída KOSDataAkce, která generuje akce v rozvrhu pro celý týden od pondělí do neděle. Toto generování se provádí až do předposledního týdne včetně.

Poslední týden obvykle opět nebývá celý od pondělí až do neděle. Z datové struktury obsahující data získaná za pomoci analyzátoru jsou vybrána pouze data, která jsou potřebná pro generovaní posledního týdne. Je určen typ dne v rámci týdne pro koncové datum importu rozvrhu. Následně jsou vybrána data od počátku týdne (pondělí) až do posledního dne období, na které je rozvrh importován. Poté se provede generovaní akci pomocí třídy KOSDataAkce. Pokud import rozvrhu z KOSu je proveden úspěšně, je uživatel informován o úspěšném importu rozvrhu z KOSu.

### 6.6.4.2 Vytvoření dat pro Víceuživatelský plánovací kalendář

Data jsou dále zpracována pomocí třídy KOSDataAkce. Postupně jsou vybírány položky z datové struktury, ve které jsou data dodaná třídou SyntaktickyAnalyzator. Nejprve zjistím zda jde o klíčové slovo označující týden, určeno klíčovými slovy „lichý“ či „sudý“. Pokud ano, uložím si tuto informaci a nastavím příznak, že proběhla změna týdne. Pokud následující datová položka je opět klíčové slovo označující týden, pak jde o rozvrh, který je v sudý i lichý týden stejný a není třeba vytvářet zvlášť rozvrh pro lichý a sudý týden.

Následně testuji zda data obsahují klíčové slovo označující den (po, pá, st, čt, pá, pondělí, úterý, středa, čtvrtek, pátek, sobota, neděle, mo, tu, th ,we, th, fr). Pokud je položka klíčové slovo označující den, tak upravím datum (den, měsíc, rok), které je následně použito pro určení data jednotlivých akcí v rozvrhu. S využitím třídy java.util.Calendar upravím datum a zkontroluji, zda se jedná o korektní datum. Následně nastavím příznak, že došlo ke změně dne a data. Pokud došlo ke změně dne, tak dojde k resetu řídících proměnných. Každý den v rozvrhu začíná v 7:30 a končí v 20:30.

Dále testuji zda data obsahují klíčové slovo „COLSPAN“. Pokud ano, tak to znamená, že jsem v rozvrhu narazil na další akci, a proto je třeba vytvořit již dříve načtenou akci a následně upravit řídící proměnné obsahující údaje o čase. Počáteční čas akce je určen koncovým časem předešlé akce. Pokud jde o první akci v daný den, tj akci od 7:30, pak je počáteční čas nastaven na 7:30. Koncový čas se musí vypočítat z počtu sloupců u klíčového slova „COLSPAN“, které reprezentuje délku probíhající akce v HTML kódu generovaném KOSem. Jednomu dni odpovídá patnáct sloupců, takže na tradiční dobu výuky, tj. 1.5 hodiny, připadají dva sloupce. Nejprve si určím počet skupin po dvou sloupcích, přičemž každé skupině odpovídá 1.5 hodiny. Pokud jde o lichý počet sloupců, tak přičtu ještě dalších 45 minut. Výsledný konečný čas určím jako součet počátečního času akce a doby po níž akce trvá.

**Konečný čas = Počáteční hodina + Počet skupin po 2 hodinách * 1.50 + Počet lichých hodin * 0.75**

Protože rozvrh generovaný KOSem nebere v úvahu přestávky mezi vyučovacími hodinami, je třeba provést korekci počátečního času o 15 minut dopředu. Poté otestuji zda jde skutečně o akci v rozvrhu nebo se jedná pouze o výplň mezi skutečnými akcemi (neobsahuje žádný text). Pokud se jedná pouze o výplň, tak upravím řídící proměnné a počáteční čas případné následující akce. Počáteční čas následující akce by byl roven koncovému času současné akce. Pokud jde o skutečnou akci, tak vytvořím novou akci a uložím ji. Parametry potřebné pro vytvoření akce:

**new Akce (String text, int priorita, int start_rok, int start_mesic, int start_den, int start_hodina, int start_minuta, int konec_rok, int konec_mesic, int konec_den, int konec_hodina, int konec_minuta, int id_akce, int id_opakovani, int id_uzivatel,String typ_akce, String typ_data, String skupina )**

Takto postupně procházím celou datovou strukturu a upravuji datum a vytvářím akce. Během jednoho projití datové struktury vytvořím rozvrh pro jeden celý týden od pondělí do pátku pro lichý i sudý týden.

## 6.7 Popis datových formátů pro uložení plánů a nastavení

Pro XML existují dva základní druhy parserů – SAX a DOM. Použití těchto parserů má obrovskou výhodu, protože nemusíme kontrolovat syntaxi dokumentu. Tu automaticky zkontroluje parser, navíc může ověřit i validitu oproti danému schématu. Naše aplikace pak nemusí obsahovat zdaleka tolik kódu pro ošetření chyb ve zpracovávaných datech. Další výhodou parseru je to, že nám obsah dokumentu zpřístupní v příjemné podobě.

Rozhraní SAX (Simple API for XML) je založeno na řízení pomocí událostí. Pomocí rozhraní vytvoříme vazbu mezi událostmi, které generuje parser, a naším kódem. V praxi to znamená, že si definujeme funkce, které se zavolají v okamžiku, kdy parser narazí na začátek elementu, na obsah elementu, na konec elementu, na komentář, na instrukce pro zpracování atd. Naší funkci jsou pak předány všechny potřebné parametry jako např. název elementu. Výhoda událostmi řízeného přístupu je v jeho rychlosti a malé spotřebě paměti. Jednotlivé události jsou vyvolávány postupně, jak je čten dokument. Při použití SAX naše aplikace bude rychlejší a bude mít malé paměťové nároky.

Rozhraní DOM (Document Object Model) je postaveno na zcela odlišném principu než SAX. Dokument je reprezentován jako stromová hierarchická struktura, kdy každému elementu odpovídá jeden uzel stromu. Odpovídající uzly mají samozřejmě i komentáře, instrukce pro zpracování atd. Tomuto způsobu reprezentace se říká grove (Graph Representation Of property Values). Rozhraní DOM obsahuje funkce, které nám umožňují celý strom dokumentu procházet, modifikovat jeho jednotlivé uzly, mazat je a přidávat. Na rozdíl od SAXu nemusíme dokument procházet od začátku do konce, ale můžeme se v něm pohybovat dle naší potřeby. Proto se rozhraní DOM uplatní v aplikacích, které provádějí náročnější operace s dokumentem.

Pro načítání plánu a nastavení aplikace z externího souboru jsem využil SAX parser. Naopak pro tvorbu dokumentů pro export plánu a úpravu souboru s nastavením aplikace jsem použil DOM parser.

### 6.7.1 Popis datového formátu pro uložení plánu

Kořenem celého datového formátu je element musc_akce, který je datového typu MUSC_AKCE. Tento element tvoří základní strukturu a zastřešuje celou datovou strukturu.

```
<xsd:element name="musc_akce" type="MUSC_AKCE"/>
```

Základní datový typ MUSC_AKCE je složený datový typ. Datový typ MUSC_AKCE je tvořen dvěma elementy složeného typu. První element je info a je datového typu INFO. Druhý element datového typu MUSC_AKCE je akce, který je datového typu AKCE. Element info se nemusí v datovém typu MUSC_AKCE vyskytovat vůbec nebo se může vyskytovat maximálně jedenkrát. Element akce se v datovém typu MUSC_AKCE nemusí opět vyskytovat ani jednou. Na rozdíl od elementu info maximální výskyt elementu akce není nijak ohraničen.

```
<xsd:complexType name="MUSC_AKCE">
  <xsd:sequence>
    <xsd:element name="info" type="INFO" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="akce" type="AKCE" minOccurs="0"
    maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Element info slouží k poskytnutí informací o programu, který daný soubor vytvořil. Základní datový typ INFO je složený datový typ. Tento datový typ se skládá ze pěti elementů, které jsou tvořeny jednoduchými datovými typy. Tyto elementy jsou – generator, verze, datum, autor a copyright. Všechny elementy jsou datového typy string. Element generator poskytuje informaci o názvu programu, který tento soubor vytvořil. Tento element se musí vyskytovat právě a pouze

jednou. Element verze poskytuje uživateli informaci o verzi programu, který tento soubor vytvořil. Element verze se musí vyskytovat právě jednou. Další element je datum, který poskytuje informaci, kdy byl tento dokument vytvořen. I tento element se musí vyskytovat přesně jedenkrát. Element autor poskytuje informaci o jménu autora tohoto programu. Element autor musí být opět přítomen přesně jedenkrát. Posledním elementem je copyright, který poskytuje informace o autorských právech. Jako ostatní elementy i tento element se musí vyskytovat právě jedenkrát.

```
<xsd:complexType name="INFO">
 <xsd:sequence>
  <xsd:element name="generator" type="xsd:string" minOccurs="1"
  maxOccurs="1"/>
  <xsd:element name="verze" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="datum" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="autor" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="copyright" type="xsd:string" minOccurs="1"
  maxOccurs="1"/>
 </xsd:sequence>
</xsd:complexType>
```

Element akce je složeného datového typu AKCE a umožňuje tvorbu jednotlivých akcí. Složený datový typ AKCE je složen celkem ze 16 elementů. Jsou to tyto elementy – start_rok, start_mesic, start_den, start_hodina, start_minuta, konec_rok, konec_mesic, konec_den, konec_hodina, konec_minuta, text, skupina, typ, priorita, opakovat_pocet a interval. Všechny položky se musí povinně vyskytovat právě jedenkrát.

Trojice elementů – text, skupina a typ jsou datového typu string. Element text obsahuje text (informační obsah) samotné akce. Aby se zabránilo možným pokusům o poškození aplikace pomocí příliš dlouhých vstupů, je z elementu text akceptováno pouze prvních 512 znaků. Element skupina obsahuje údaje o skupině do níž akce patří. U tohoto elementu je podobně jako u elementu text kontrolována délka vstupních dat. V tomto případě je délka vstupního řetězce omezena na 50 znaků. Element typ poskytuje další pomocné informace o akci, např. typ týdne u rozvrhu (lichý nebo sudý). Stejně jako u elementu skupina je zde omezena délka vstupních dat na 50 znaků.

Element priorita obsahuje číselný údaj o prioritě akce. Tento element je datového typu integer a je pro něho stanovena minimální hodnota 0 a maximální hodnota 10.

Element opakovat_pocet společně s elementem interval slouží k tvorbě pravidelně se opakujících akcí. Položka opakovat_pocet udává počet výskytů opakující se akce. Naproti tomu položka interval udává časový interval ve dnech mezi dvěma opakujícími se akcemi. Oba dva elementy jsou datového typu integer. Pokud nastane situace, že položka interval je nenulová a položka opakovat_pocet je naopak nulová, není provedeno opakování akcí a daná akce je importovaná pouze jednou.

Zbývající položky datového typu AKCE jsou datového typu integer. Tyto elementy slouží k určení počátečního a koncového času akce.

Ačkoliv samotný datový formát popsaný pomocí XSD provádí již řadu kontrol vstupních dat, jako je např. datový typ nebo rozsah hodnot, není datový formát schopen ošetřit platnost dat obsažených v jednotlivých elementech a to, že se např. koncové datum akce předchází počátečnímu. Tento problém je řešen při samotném importu akcí až při načtení všech elementů tvořících datum. Následně je správnost data zkontrolována s pomocí třídy java.util.Calendar. V rámci jednoho importu lze maximálně importovat takový počet akcí, který je povolen administrátorem na straně serveru. Tento počet je společný jak pro neopakující se akce, tak i pro opakující se akce.

### 6.7.2 Popis datového formátu pro uložení nastavení aplikace

Kořenovým elementem je musc_options, který je datového typu MUSC_OPTIONS. Složený datový typ MUSC_OPTIONS obsahuje dva elementy – view_options a priority_options

```
<xsd:complexType name="MUSC_OPTIONS">
  <xsd:sequence>
    <xsd:element name="view_options" type="VIEW_OPTIONS" minOccurs="0"
    maxOccurs="1"/>
    <xsd:element name="priority_options" type="PRIORITY_OPTIONS"
    minOccurs="0" maxOccurs="11"/>
  </xsd:sequence>
</xsd:complexType>
```

Element view_options je datového typu VIEW_OPTIONS. Tento element slouží k uložení globálních vlastností aplikace např. barva pozadí atd. Složený datový typ VIEW_OPTIONS se skládá z pěti elementů – header, background, selected_day, today a one_day_action_month. Každý z těchto elementů se nemusí vyskytovat vůbec nebo se může vyskytovat maximálně jedenkrát. Všechny tyto elementy jsou složeného datového typu, který obsahuje element color. Element header slouží k nastavení barvy záhlaví jednotlivých pohledů. Element background umožňuje nastavit barvu pozadí v jednotlivých pohledech. Dalším elementem je selected_day, který nastavuje barvu pro zvýraznění dne v Týdenním plánu při výběru dne v Měsíčním pohledu. Element today poskytuje možnost určit barvu pro zvýraznění dnešního dne. Posledním elementem je one_day_action_month, který umožňuje nastavit podkladovou barvu v Měsíčním plánu pro jednodenní akce.

```
<xsd:complexType name="VIEW_OPTIONS">
  <xsd:sequence>
    <xsd:element name="header" type="HEADER" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="background" type="BACKGROUND" minOccurs="0"
    maxOccurs="1"/>
    <xsd:element name="selected_day" type="SELECTED_DAY" minOccurs="0"
    maxOccurs="1"/>
    <xsd:element name="today" type="TODAY" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="one_day_action_month" type="ONE_DAY_ACTION_MONTH"
    minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

Element priority_options je datového typu PRIORITY_OPTIONS. Tento složený datový typ slouží k nastavení barev pro zobrazení jednotlivých priorit. Program rozeznává až 11 různých priorit. Z tohoto důvodu se element priority_options nemusí vyskytovat vůbec nebo se může vyskytovat maximálně 11. Datový typ PRIORITY_OPTIONS obsahuje dva elementy – priority a color. Element priority určuje číslo priority pro níž má být určena barva. Je datového typu integer a může nabývat hodnot od nuly do desíti. Element color slouží k určení barvy pro jednotlivé priority.

Element color je datového typu COLOR. Tento složený datový typ slouží k uložení barvy v barevném modelu RGB. Datový typ COLOR obsahuje tři elementy, které určují jednotlivé složky barevného modelu – red, green a blue. Tyto elementy se musí vykytovat vždy právě jedenkrát. Každý z těchto elementů je datového typu integer a nabývá hodnoty od 0 do 255.

## 6.8 Bezpečnost programu MUSC

Jedním ze základních požadavků kladených na program MUSC byla bezpečnost a zajištění ochrany osobních údajů. Osobními údaji jsou zde myšleny nejen osobní údaje zadané při vytvoření účtu, ale i údaje obsažené ve vlastních záznamech plánovaných událostí. Zabezpečení programu MUSC lze rozdělit do dvou částí. Za prvé je to zabezpečení přístupu do systému, které je řešeno uživatelským jménem a heslem. Za druhé je to bezpečnost dat přenášených po počítačové síti, což je řešeno šifrováním.

### 6.8.1 Zabezpečení přístupu do systému

Při spuštění má uživatel možnost si zvolit ze dvou módů přístupu do programu:

1. Uživatel zadá své uživatelské jméno a heslo. Poté jsou tyto údaje ověřeny a pokud jsou správné, je mu umožněn vstup do programu. V opačném případě je uživatel vyzván k opětovnému zadání přihlašovacích údajů. Po úspěšném zadání přihlašovacích údajů má uživatel možnost plně editovat svůj plán (vytvářet nové akce, opravovat akce a mazat akce). Pokud se přihlášený uživatel rozhodne prohlížet plány ostatních uživatelů, musí nejprve zadat uživatelské jméno požadovaného uživatele. Pokud daný uživatel existuje, provede se průnik skupin do nichž patří přihlášený uživatel a uživatele, který má být zobrazen (průnik se nevztahuje na skupinu PRIVATE). Následně jsou zobrazeny akce, které patří do skupin obsažených v průniku. Akce, které náleží cizímu uživateli, si může přihlášený uživatel pouze prohlížet a nemůže je nijak editovat (vytvářet nové akce, opravovat akce a mazat akce).

2. Uživatel zvolí volbu anonymní host a zadá uživatelské jméno osoby jejíž plán chce zobrazit. Pokud daná osoba existuje, je nepřihlášený uživatel vpuštěn do aplikace. Následně jsou zobrazeny pouze akce, které patří do skupiny ALL (registrovaný uživatel tak označuje běžné události, které mohou být viditelné pro ostatní uživatele) a skupiny KOS (do skupiny KOS náleží školní rozvrh). Akce, které náleží cizímu uživateli, si může nepřihlášený uživatel pouze prohlížet a nemůže je nijak měnit (vytvářet nové akce, opravovat akce a mazat akce).

### 6.8.2 Šifrování

V současné době je kladen velký důraz na bezpečnost a ochranu osobních údajů. V rámci programu MUSC dochází k přenosu osobních údajů o uživateli, jako je jméno a příjmení, heslo, e-mail atd., přes síť Internet. Je zde proto reálné nebezpečí získání těchto údajů a jejich následné zneužití. Z tohoto důvodu bylo rozhodnuto, že přenos dat po síti v rámci programu MUSC musí být zabezpečen. Rozhodl jsem se pro implementaci symetrické šifry z důvodu její snadné implementace, vysoké rychlosti šifrování a dešifrování a především pro její vysokou bezpečnost.

#### 6.8.2.1 Symetrická šifra

V současné době slouží k rychlému a utajovanému přenosu většího objemu dat symetrické šifry. Symetrické šifry jsou často používány ve spojení s asymetrickými šiframi. Asymetrické šifry jsou použity pro autentizaci a předání klíče pro symetrickou šifru. Další přenos dat je pak šifrován pomocí symetrické šifry. Důvodem pro nepoužívání asymetrické šifry po celou dobu komunikace je skutečnost, že její výpočet je 10krát až 100krát pomalejší než u symetrické šifry.

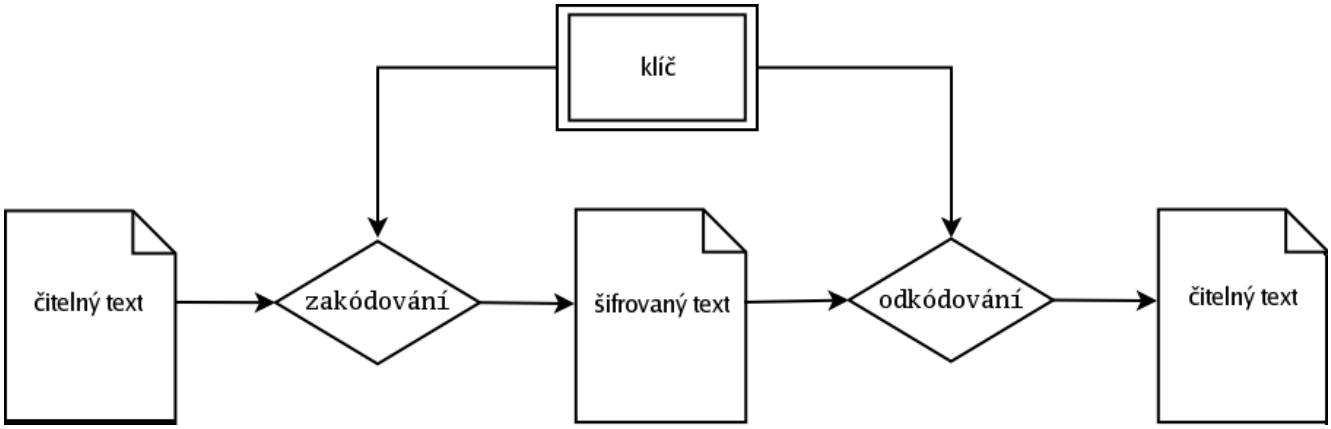
**Definice: Symetrický kryptografický systém pro šifrování zpráv (symetrická šifra)** je taková šifra, kde pro každé k, kde $k \in K$, lze z transformace zašifrováním $E_k$ určit transformaci dešifrování $D_k$ a naopak.

Z výše vyplývající symetrie jsou tyto systémy označované jako symetrické systémy a jejich klíče jako symetrické klíče. Symetrické klíče jsou tajné, ale jak zobrazení E, tak zobrazení D, mohou být zcela veřejná. Symetrická šifra, nazývaná také konvenční šifra, je založena na principu jednoho klíče, který slouží jak k zašifrování dat, tak i k jejich následnému dešifrování. Symetrické kódy mají jako hlavní výhodu velkou rychlost algoritmu šifrováni a dešifrování. Na druhou stranu je však potřeba zajistit, aby se příjemce i odesílatel dohodli na jednom společném klíči, který budou znát pouze oni dva a už nikdo další. Problémem je tedy distribuce klíče, jak dostat klíč od odesílatele k příjemci aniž by ho získal někdo nepovolaný. V programu MUSC je tento problém řešen tak, že klíč je součástí zdrojových kódů aplikace, jak na straně klienta, tak i na straně serveru. Nedochází zde proto k přenosu symetrického klíče po síti. Postup při šifrování dokumentů je zachycen na obrázku 11.

Symetrické šifrovací algoritmy lze rozdělit do dvou skupin: proudové a blokové šifry. Proudové šifry kódují jednotlivé byty bit po bitu. Naopak blokové šifry vezmou určitý předem daný objem dat a zakódují ho jako jeden celek. V současné době bývají tyto bloky obvykle dlouhé 64 bitů. Dnes je snaha používat bezpečné algoritmy což pro blokové šifry znamená:

1. Používat ty algoritmy, které mají dostatečnou délku klíče. Za tu se dnes považuje délka 64 až 100 bitů.

2. Opírat se o algoritmy konstruované tak, aby byly odolné proti pokusům typu brute force, statistickým útokům a potenciálním kryptoanalytickým metodám. Význačnými představiteli takovýchto prostředků jsou diferenciální a lineární kryptoanalýzy.




**Obrázek 11: Postup při šifrování dokumentů**

### 6.8.2.2 Symetrická šifra Blowfish

Šifra Blowfish byla navržena Bruce Schneierem, který ji poprvé zveřejnil roku 1993. V současnosti se tato šifra využívá v řadě průmyslových produktech k ochranně citlivých údajů. Algoritmus Blowfish byl již od počátku zamýšlen jako náhrada k algoritmu DES. DES je licencovaný algoritmus, zatímco šifrovací algoritmus Blowfish je nepatentovaná, nelicencovaná a bez copyrightu alternativa k DESu. Ale největší nevýhodou algoritmu DES je to, že se ho již podařilo prolomit metodou hrubé síly v relativně přijatelných časech. Na rozdíl od DESu byl

Blowfish již od svého počátku poskytnut veřejnosti a měla k němu přístup celá řada kryproanalytiků. I přes tento fakt není do dnešních dní znám žádný efektivní způsob prolomení tohoto algoritmu. Jedná se o symetrickou blokovou šifru s délkou bloku 64 bitů a klíčem délky od 32 bitů do maximálně 448 bitů tj. 56 Bytů (obvykle však 128 bitů).

Rozhodl jsem se pro implementaci šifry Blowfish. Tuto šifru jsem si vybral z několika důvodů:

1. Pro její vysokou bezpečnost. Do počátku roku 2006 nebyl znám žádný efektivní způsob prolomení této šifry.

2. Protože se jedná o symetrickou šifru, je provádění šifrování a dešifrování velmi rychlé.

3. Tento algoritmus je od Javy verze 1.5 její standardní součástí. Přestože samotný algoritmus je již sám o sobě velmi rychlí, věřím že je zde velmi efektivně implementován již s ohledem na jazyk Java, což by mohlo přinést ještě větší efektivitu.

4. Tato šifra není licencovaná a lze ji volně používat.

### 6.8.2.3 Popis algoritmu Blowfish

Algoritmus je tvořen dvěmi částmi: část expanze klíče a část šifrování dat. Expanze klíče převádí klíč s libovolnou délkou od 32 bitů do 448 bitů tj. maximálně 56 Bytů na několik polí podklíčů, které dávají dohromady celkem 4168 Bytů. Šifrování dat je prováděno po blocích dlouhých 64 bitů v šestnácti rundách. Každá runda provádí permutaci závislou na klíči a substituci závislou jak na kódovaných datech tak i klíči. Všechny operace použité v algoritmu jsou XOR anebo sčítání 32 bitových slov. Navíc jsou v každé rundě prováděny 4 operace výběru dat z pole vypočteného indexu.

#### 6.8.2.3.1 Podklíče

Pro Blowfish je typické, že při své činnosti používá velký počet podklíčů, které musí být vypočteny ze zadaného původního klíče ještě před samotným šifrováním, resp. dešifrováním dat. Podklíče jsou uloženy celkem v pěti polích. První pole, označované jako P-pole anebo P-box, obsahuje celkem osmnáct 32 bitových položek, které jsou dále označovány jako P1, P2, … , P18. Ostatní pole jsou označována jako S-pole nebo také jako S-boxy. Každý S-box má dvě stě padesát šest 32 bitových položek. Pokud budeme pracovat s S-boxem i, kde i = 1, 2, 3, 4, pak jednotlivé položky budeme postupně označovat $S_i0$, $S_i1$,…, $S_i255$. Je to vlastně Feistelova šifra s 16 rundamï a využívá velké závislosti na S-boxech. Tato struktura je podobná algoritmu CAST-128, který využívá S-boxy pevné délky.

#### 6.8.2.3.2 Šifrování dat

Postup při šifrování u algoritmu Blowfish je ukázán na obrázku 12. Vstupem, který označujme X, je slovo dlouhé 64 bitů, které se rozdělí na dvě části (levou L a pravou R část). Přičemž každá část je dlouhá 32 bitů. Levou část označíme xL a pravou xR (xL zahrnuje bity 32 : 63, xR pak obsahuje bity 0 : 31 ze vstupu X). Šifrování probíhá v 16ti rundách, což lze popsat následujícím algoritmem.

**FOR** i = 1 **TO** 16 **DO**

**BEGIN**

xL = xL **XOR** Pi

xR = F (xL) **XOR** xR

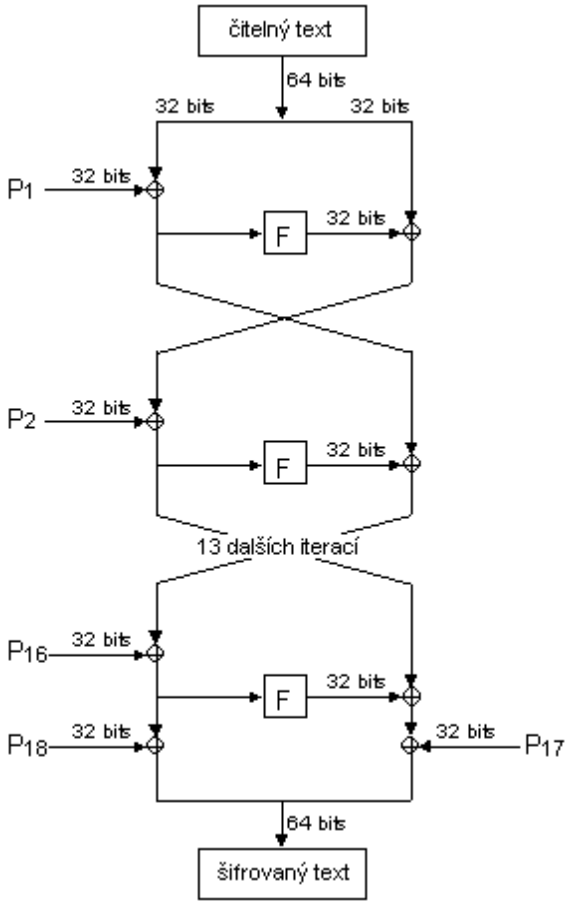
**SWAP** (xL, xR) **/* provede záměnu obou parametrů */**

**END;**

Po tomto cyklu následuje ještě tato série příkazů:

**SWAP** (xL, xR) **/* odstraní poslední swap v cyklu */**

xR = xR **XOR** P17
xL = xL **XOR** P18

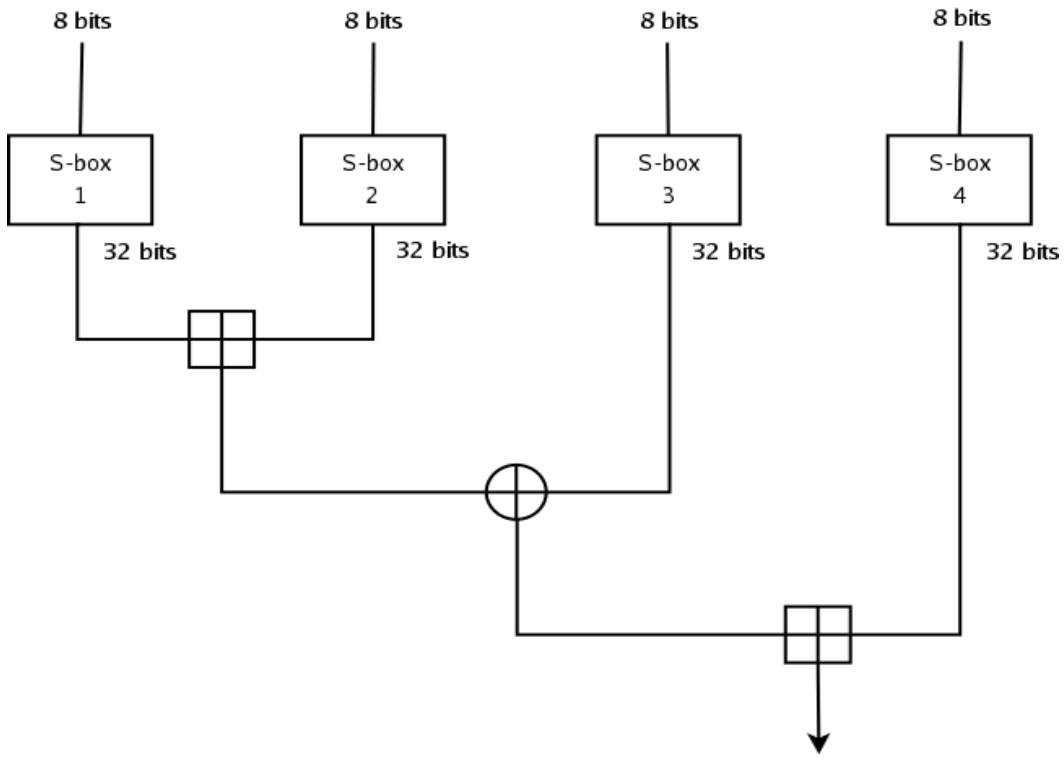



**Obrázek 12: Algoritmus šifrování u algoritmu Blowfish**

Posledním krokem je spojení části xL a xR do výstupního 64 bitového bloku zašifrovaného textu. Funkce F rozdělí 32 bitové vstupní slovo Y na 4 samostatné části a, b, c, d, každé po 8 bitech. Takto získané části v řadě po sobě představují jednotlivé byty (tj. 8 bitů) vstupního slova zleva doprava (tj. část **a** představuje bity 24 : 31, část **b** bity 16 : 23, část **c** bity 8 : 16 a část **d** bity 0 : 7) a používají se jako indexy do S-boxů. Výstupní hodnota funkce je pak definována podle následující funkce:

**F (y) = ((S1, a + S2, b mod 232) XOR S3, c) + S4, d mod 232**

Obdobným způsobem jako probíhá šifrování je realizováno i dešifrování, ale s tím rozdílem, že položky P-boxu (tj. $P_i$ , kde i = 1, 2, ... , 18) jsou použity v opačném pořadí než při šifrování. Princip funkce F (y) je ukázán na obrázku 13.




**Obrázek 13: Principielní schéma funkce F (y)**

### 6.8.3 Zabezpečení poznámek

Pro každou kopii programu MUSC lze vytvořit jeden soubor pro ukládání poznámek. Tento soubor je společný vždy pro všechny uživatele používající danou kopii programu. Tento soubor je uložen na lokálním disku klientské aplikace. Přístup do tohoto souboru lze vyvolat pomocí položky „Moje Poznámky (To Do List)“ v menu Soubor. Tato funkce slouží k tvorbě krátkých poznámek bez nutnosti jejich spojení s konkrétním časovým okamžikem. Protože je tento soubor uložen na lokálním disku klienta, hrozí zde případná možnost zneužití. Z tohoto důvodu je i tento soubor šifrován pomocí algoritmu Blowfish.

### 6.8.4 Zabezpečení hesla v databázi

K datům uloženým v databázi má přístup pouze administrátor. Naneštěstí nelze nikdy vyloučit selhání lidského faktoru nebo útok za účelem získání osobních údajů registrovaných uživatelů. Pravděpodobně nejcitlivějším údajem je pak heslo pro přístup do systému. Z tohoto důvodu je heslo, předtím než je při vytvoření nového účtu uloženo do databáze, šifrováno. I zde je využit šifrovací algoritmus Blowfish. Z důvodu bezpečnosti je zde použit jiný klíč pro šifrování a dešifrování, než je použit pro šifrování údajů přenášených po síti.

### 6.8.5 Kontrola vstupů

Problematika korektních vstupů je jednou ze základních otázek u každé aplikace komunikující s uživatelem. Nejedná se zde však pouze o syntaktickou správnost vstupů (např. očekáváme číslo a dostaneme text, ale také logickou správnost (př. koncový den akce nesmí předcházet počátečnímu) a v neposlední řadě i kontrola s ohledem na příliš dlouhé nebo chybějící vstupy. Pokud dojde k jakémukoliv problému s ohledem na výše zmíněné možnosti, je uživatel na tuto skutečnost upozorněn s co největším důrazem na určení příčiny chyby a ne až jejích následků.

Pokud nějaký povinný vstup chybí je uživatel na tuto skutečnost upozorněn. V případě kontroly vstupů s ohledem na jejich délku, jsou všechny vstupy omezeny na určitý počet znaků. Všechny vstupy, kromě vstupů určujících text u události, akceptují maximálně 50 znaků. Pokud je vstup delší než 50 znaku, jsou všechny znaky nad 50 ignorovány. U vstupu představující text akce je maximální možný počet akceptovaných znaků stanoven na 512 znaků. Všechny znaky nad 512. znak jsou ignorovány.

Pokud vstup představuje prostý text, tak u něho není prováděna syntaktická kontrola. Pokud však má být vstup číslo, je u něho provedena syntaktická kontrola vždy. Jestliže číslo neprojde testem, je uživatel opět na tuto skutečnost upozorněn. Speciální kategorií pro kontrolu vstupů jsou data a časové údaje. Jak čas, tak i data mají, při vstupu v dialogu pro tvorbu nových akcí předepsanou syntaxi. Pro čas je tato syntaxe hh:mm, kde h je hodina a m je minuta. Pro datum je určena tato syntaxe DD.MM.RRRR, kde D představuje den, M je měsíc a R je rok. Pro oba případy, jak čas tak i datum, platí, že pokud některá z položek h, m, D, M nebo R je v intervalu od 0 do 9, pak není nutné tento údaj doplňovat nulou. Jak pro časové vstupy, tak i pro vstupy dat, je nejprve provedena kontrola jejich vstupní syntaxe. Pokud proběhne kontrola u obou položek korektně, přistoupí se následně k logické kontrole.

Zde je potřeba zkontrolovat nejen to, že údaj nepřesahuje stanovené rozsahy (např. měsíc má rozsah od 1 do 12), ale také zda vložené datum je platné z pohledu kalendáře (např. přestupný rok). Tato kontrola je provedena s využitím třídy java.util.Calendar. Poté co je zkontrolována možná existence data, je ještě potřeba provést kontrolu, zda náhodou koncové datum akce nepředchází počátečnímu.

### 6.8.6 Styly přihlášení

Registrovaný uživatel má možnost přihlásit se do sytému pomocí dvou základních stylů. První styl je tzv. „klasické přihlášení“ a druhý styl je tzv. „rychlé přihlášení“.

1. „Klasické přihlášení“ představuje přihlášení pomocí tradičního přihlašovacího dialogu. V tomto dialogu je uživatel vyzván k zadání svých identifikačních údajů, jako je uživatelské jméno a heslo. Následně dojde k odeslání požadavků pro ověření identity uživatele na server. Pokud jsou identifikační údaje neplatné, je uživatel vyzván k opětovnému zadání identifikačních údajů. Pokud jsou však identifikační údaje správné je uživatel vpuštěn do programu a je mu umožněna práce s jeho plánem.

2. Druhým stylem je tzv. „rychlé přihlášení“. Tento styl přihlášení je možný pouze při startu aplikace MUSC. Po vytvoření účtu v systému a následném prvním přihlášení, má uživatel možnost uložit si své identifikační údaje pomocí položky „Uložit heslo“ v menu Soubor. Tato možnost uložení identifikačních údajů zůstává i při každém dalším přihlášení. Pokud uživatel následně spustí program není již vyzván k zadání identifikačních údajů. Identifikační údaje jsou získané ze souboru, do kterého byly již dříve uloženy, a je přímo vyslán požadavek na server na ověření identity uživatele. Stejně jako u „klasického přihlášení“ pokud jsou identifikační údaje neplatné, je uživatel vyzván k jejich opětovnému zadání pomocí přihlašovacího dialogu. Jestliže jsou údaje správné, je uživateli umožněna práce s jeho plánem.

   Protože tato funkce by mohla znamenat významné bezpečností problémy, byla přijata

řada bezpečnostních opatření:

a) Uživatel je vyzván aby zadal jméno souboru do něhož jsou identifikační údaje uloženy. Na jméno souboru nejsou kladeny žádné požadavky. Z této skutečnosti tedy vyplývá fakt, že soubor může být např. pojmenován „obrazek.png“. Následně se tento soubor může tvářit jako poškozený soubor s obrázkem.

b) Identifikační údaje jsou uloženy na lokální disk klienta v zašifrované podobě. K šifrování je použit algoritmus Blowfish.

c) Souboru s identifikačními údaji může být umístěn na jakémkoliv místě v počítači či na přenosném médiu. Představou je, že uživatel bude mít soubor s identifikačními údaji uložen např. na flashdisku. Tento flashdisk před začátkem práce s aplikací připojí k počítači a poskytne, tak aplikaci MUSC soubor s identifikačními údaji. Po skončení práce s aplikací a jejím ukončení, odpojí uživatel flashdisk s identifikačním souborem. Při takovéto práci s aplikací a používáním souboru s identifikačními údaji se významně snižuje riziko zneužití souboru s identifikačními údaji cizí osobou, která přijde k počítači na němž funguje aplikace MUSC a pokusí se připojit do systému.

Protože název souboru i umístění souboru s identifikačními údaji může být libovolné, je třeba specifikovat cestu k tomuto souboru. Aby byl program schopen určit polohu souboru s identifikačními údaji, je potřeba mu tuto polohu souboru specifikovat jako parametr při spuštění s využitím souboru MUSCclient.bat. Snadnou editací tohoto souboru, uživatel zadá buď absolutní nebo relativní cestu k souboru s identifikací. Další informace o editaci a struktuře souboru MUSCclient.bat naleznete v uživatelské příručce na přiloženém DVD. Jak již bylo nastíněno výše, předpokládá se, že uživatel bude mít soubor s identifikací na flashdisku a z tohoto důvodu se absolutní adresa umístění souboru po připojení disku na stejném počítači nebude měnit. Proto lze provést snadnou editaci souboru pouze jednou při prvním vytvoření souboru s identifikačními údaji.

Tato funkce do jisté míry oslabuje zabezpečení systému, ale je však pouze na uživateli, jak se k této funkci postaví, ať již z pohledu využití této možnosti, tak i z pohledu bezpečné práce se souborem s identifikačními údaji.

## 6.9 Sledování funkčnosti aplikace

Během vývoje celé aplikace probíhalo testování všech strukturních celků a funkčních bodů aplikace. V konečné fázi se pak přidalo ještě uživatelské, výkonnostní a systémové testování. Ačkoliv probíhalo důkladné testování během celého vývoje aplikace, je možné, že by se v aplikaci mohly v budoucnu vyskytnout nějaké chyby. Protože hledání a opravování chyb s dlouhým časovým odstupem od vývoje samotné aplikace může být zdlouhavé, byla do aplikace zabudována podpora pro sledování chování aplikace a detekci případných chyb. Z tohoto důvodu byly všechny potenciálně kritické sekce, kde by se mohla vyskytnout chyba, ošetřeny proti chybám. Toto ošetření obvykle nastaví defaultní hodnoty proměnných u nichž chyba nastala a provede záznam o výskytu chybného stavu. Tyto údaje jsou zaznamenávány do externího souboru. V tomto souboru se vždy nachází informace o všech vzniklých chybách v rámci posledního spuštění aplikace. V případě výskytu nějaké chyby může pak uživatel, kromě svého ústního popisu chyby, připojit i tento soubor k informaci o problémech s aplikací.

Při práci s GUI uživatel o chybové výstupy na konzoli přichází a jediné co může zaregistrovat je nestandardní chování aplikace bez bližších informací. Z tohoto důvodu byl do souboru s chybovými informacemi přesměrován chybový výstup aplikace. Tento soubor se nachází jak na straně klientské aplikace, tak i na straně serveru. Na straně klienta je to soubor ClientMUSCerror.log a na straně serveru je to soubor ServerMUSCerror.log.

# 7 Testování

Během vývoje aplikace probíhaly čtyři typy testů. Prvním typem testu byly testy funkčnosti aplikace. Druhým typem testů byly uživatelské testy, třetím byly výkonnostní testy a posledním typem byly testy systémové.

## 7.1 Funkční testování

Funkční testy probíhaly ve dvou fázích. V první fázi probíhalo tzv. white-box testování a v druhé fázi tzv. black-box testování. White-box testování (strukturální testování) se zaměřilo na skutečnost zda všechny vnitřní komponenty správně fungují. Toto testování probíhalo již během samotného vývoje aplikace. Pro každou nově vzniklou funkcionalitu programu byla vytvořena množina testovacích dat. Důraz byl kladen především na kontrolu všech cyklů a řídících struktur. Každá řídící strukturu byla provedena ve všech možných větvích. Dále byly testovány všechny možné kombinace podmínek v řídících strukturách.

V druhé fázi funkčních testů probíhalo black-box testování (funkcionální testování), které se zaměřilo na správné fungování všech navržených funkcí programu. Toto testování probíhalo také již při samotném vývoji. Testy se prováděly vždy po vytvoření nového funkčního bodu a při jeho integraci do již existující aplikace. Testování se především zaměřilo na nestandardní činnosti s daným funkčním bodem a na kontrolu vstupních dat. V rámci nestandardního chování se např. testovalo neřízení se pokyny udanými aplikací nebo pokus o připojení k serveru na adrese, na které server neběžel apod.

Kontrola vstupů se pak zabývala zejména hraničními hodnotami vstupů. V tomto případě se jednalo ať již o prázdné vstupy nebo hodnoty mimo rozsah povolených vstupů. Dále se také testovala odolnost aplikace např. na příliš dlouhé vstupy nebo špatný formát vstupních dat, místo požadovaných čísel byl zadán text apod. Protože aplikace pracuje s časy a daty, bylo nutné provést testy i na tuto oblast. U časů se testy zaměřily opět na krajní hodnoty. U data se testy zaměřily především na jejich platnost s ohledem na kalendář a na pořadí dat vzhledem k počátku a konci událostí.

## 7.2 Uživatelské testování

Uživatelské testy se zaměřily na dvě problematiky. První bylo správné fungování GUI aplikace a druhou pak práce uživatelů s aplikací. Během testování GUI se testovalo např. zda je aktivní okno vysvícené, funkčnost klávesových zkratek nebo správná reakce na tlačítko pro zobrazení dalších voleb u tvorby akcí apod. Zároveň se testovalo, zda při pohybu myši dochází k aktualizaci všech požadovaných údajů, jako je např. aktuální čas.

V rámci práce uživatelů s aplikací se testy zabývali např. intuitivností tvorby plánů nebo rychlostí vytvoření nové akce. Důležitým faktorem byla také přehlednost jednotlivých plánů. Dále pak také možnost získání potřebných informacích o událostech. Testoval se také vzhled aplikace a barevné schéma aplikace. Na základě požadavků od testerů byl např. pozměněn vzhled některých plánů nebo také základní barevné schéma aplikace. Také původní dialog pro tvorbu akcí byl na základě požadavků zjednodušen a některé možnosti volby nastavení byly přesunuty do volitelných možností. Jedním ze vznešených požadavků byla také možnost určit datum pomocí jednoduchého kalendáře.

## 7.3 Výkonnostní testování

Výkonnostní testy byly provedeny na notebooku Prestigio Nobile 157. Hardwarová a softwarová konfigurace:

CPU: Intel Pentium M 1.7 MHz
RAM: 512 MB
Grafická karta: ATI Mobile Radeon 9700, 64 MB RAM
HD: 60 GB, 5400 ot./min
OS: MS Windows XP Pro SP2
Java: JRE 1.5_06

Výkonnostní testy se zaměřily na zpracování velkého objemu dat. Vždy se porovnávalo snížení výkonnosti při práci s 20 000 akcemi vůči 1 akci. Akce pro test měly vždy ponechané defaultní hodnoty nastavené programem. Nejprve se testoval čas na vytvoření 1 resp. 20 000 akcí (jednalo se o opakující se akce po 1 dni). Ačkoliv nárůst zatíženosti byl 20 000 krát větší, došlo pouze k 8 násobnému poklesu výkonnosti. V tento okamžik byly akce uloženy zatím ve vyrovnávací paměti. Nyní se velmi kladně projevila implementace double-bufferingu, protože při práci v rámci aktuálního pohledu nedošlo k žádnému výkonnostnímu poklesu. Při přechodu v rámci aktuálního plánu vpřed nebo vzad, došlo přibližně k 2 násobnému poklesu výkonnosti.

Dalším testem pak bylo uložení 1 respektive 20 000 akcí do databáze. Uložení 20 000 akcí trvalo přibližně asi 20krát déle oproti uložení 1 akce. Ačkoliv se jedná o velký pokles je tento pokles akceptovatelný s přihlédnutím ke skutečnosti, že třebaže požadavek vzrostl o 4 řády, nastal pokles pouze o 1 řád. Pravděpodobně se zde projevila pomalejší spolupráce s databází, která by mohla být v budoucnosti více optimalizována.

Na základě testování bylo optimalizováno šifrování použité během přenosu dat. K inicializaci šifer nyní dochází pouze jednou a to při startu aplikace. Dále bylo optimalizováno zpracování požadavků na serveru, ať již se jedná o identifikaci požadavků nebo o práci s databází.

## 7.4 Systémové testování

Systémové testování testovalo program MUSC v kombinaci s ostatními systémovými prvky, jako jsou různé operační systémy, spolupráce s databází atd. Tyto testy se zaměřily hlavně na ověření multiplatformnosti aplikace. Testy proběhly na třech rozdílných systémových konfiguracích:

1. Klientská aplikace běžela pod OS MS Windows XP Pro SP2, serverová aplikace a databázový server běžely taktéž pod OS MS Windows XP Pro SP2.

2. Klientská aplikace běžela pod OS Windows XP Pro SP2, naproti tomu serverová aplikace a databázový server běžely pod OS Linux.

3. Klientská aplikace byla spuštěna pod OS Solaris 9, naopak serverová aplikace a databázový server byly spuštěny pod OS Linux

Testy na všech použitých platformách proběhly úspěšně bez výskytu nějakých závažných chyb.

# 8 Srovnání s existujícími řešeními

V úvodní části byla provedena rešerše dvou již existujících produktů. Jednalo se o program Microsoft Outlook 2000 od společnosti Microsoft a o program Calendar Quick v3.1 od Briana K. Holdswortha.

Pravidlem pro tento typ aplikace se stalo poskytnout minimálně tři klasické pohledy na časový plán – denní, týdenní a měsíční. V tomto ohledu aplikace MUSC tyto produkty překonává, jelikož nabízí hned pět rozdílných pohledů. Prvním je klasický Denní plán. Druhým pak je Detailní denní plán, který slouží pro přesné plánování až po 15 minutových blocích. Dalším pohledem je KOS plán, který představuje klasický pracovní týden od pondělí do pátku. Poté následuje klasický Týdenní plán a na závěr je tu ještě Měsíční plán.

Obě aplikace umožňují jistým způsobem třídit akce do skupin. V tomto případě nešlo přímo přistoupit na model zvolený oběma předchozíma aplikacema. Důvodem byla skutečnost, že na rozdíl od srovnávaných aplikací je program MUSC síťovou aplikací, která umožňuje navíc sdílení plánů, které jsou uloženy v centrální databázi. Dále bylo třeba zachovat bezpečnost při sdílení, a proto uživatel nemá právo vytvářet vlastní skupiny. Z tohoto důvodu uživatel může akce pouze třídit do přidělených skupin. Tuto skupinu však může uživatel kdykoliv pro danou akci změnit. Každý uživatel hned od vytvoření náleží do tří základních skupin – PRIVATE, ALL a KOS. Do dalších skupin může být později přidán správcem databáze. Způsob jakým jsou ve srovnávaných aplikacích využity skupiny, je u MUSC nahrazen možností přiřadit události prioritu.

Stejně jako srovnávané aplikace umožňuje MUSC opakování akcí. Na rozdíl od nich však nejsou připraveny předem možné typy, ale uživatel si definuje buď počáteční a koncový čas a rozestupy mezi opakovanými akcemi, nebo při importu zadá kolikrát chce danou akci opakovat a s jakými rozestupy.

Aplikace MUSC podobně jako Outlook poskytuje tvorbu poznámek bez nutnosti jejich pevného svázání s časovým okamžikem. Podobně jako u obou aplikací i zde je stanovena nejmenší časová jednotka pro zobrazovanou akci. U Outlooku je to 30 minut a u Calendar Quick 5 minut. Obě varianty mi přišli částečně nevyhovující, a proto jsem jako nejmenší časovou jednotku zvolil 15 minut. Tato skutečnost však nezabraňuje vytvářet akce s přesnými časovými hranicemi až do přesnosti na minuty.

Jak již bylo zmíněno v rešerši, obě aplikace měli problémy s kontrolou vstupních dat, což v některých situacích vedlo k neočekávanému chování. U aplikace MUSC byla kladena velká pozornost na bezpečnost. Zejména pak na kontrolu vstupních dat, která by měla odhalit veškerá špatná vstupní data. Dále je také zajištěna bezpečnost pro komunikaci po síti. V tomto ohledu MUSC používá šifrování s využitím algoritmu Blowfish.

Aplikace MUSC se zaměřila zejména na jednoduchost a intuitivnost ovládání. Byl kladen důraz na snadnou a rychlou tvorbu plánu, bez nutnosti nastavování celé řady parametrů. Z těchto důvodů lze program doporučit začínajícím uživatelům. Program ale také nabízí celou řadu možností pro nastavení rozličných parametrů pro jednotlivé akce. Dále program umožňuje import rozvrhu z KOSu a další pokročilé funkce. Z tohoto důvodu lze program doporučit i pokročilým uživatelům, kteří si rádi specifikují své požadavky na plánováni co nejpřesněji.

# 9 Závěr

Výsledkem této bakalářské práce je aplikace Víceuživatelský plánovací kalendář. Tento program slouží k tvorbě časových plánů. Samotný program je tvořen ze tří částí – klientská aplikace, serverová aplikace a databáze. Klientská aplikace umožňuje snadné a rychlé plánování akcí a schůzek. Ovládání aplikace bylo navrženo tak, aby bylo co nejvíce intuitivní a umožňovalo snadnou a rychlou práci s programem. Časový plán může být zobrazen pomocí až pěti rozdílných pohledů – Denní plán, Detailní denní plán, KOS plán, Týdenní plán a Měsíční plán. Tyto jednotlivé pohledy umožňují přehledný, detailní a rychlý náhled na časový plán uživatele. Aplikace dále umožňuje zobrazení více současně probíhajících událostí.

V rámci tvorby akcí lze vytvářet jak jednodenní akce, tak i vícedenní. Zároveň lze vytvářet opakující se akce. Součástí aplikace je také HTML parser, který je využit při importu rozvrhu z KOSu do aplikace, skrze HTML soubor generovaný samotným KOSem. HTML parser slouží k rozdělení HTML souboru s rozvrhem na jednotlivé lexikální elementy. Z těchto elementů je následně vytvořen rozvrh v aplikaci. MUSC umožňuje tvorbu plánu dvěma rozdílnými způsoby. Prvním způsobem je vytvoření akce přímo v aplikaci s využitím myši. Druhým způsobem je pak vytvoření akcí mimo aplikaci a jejich následný import do aplikace skrze XML soubor. Aplikace také umožňuje exportovat plány jednotlivých uživatelů z aplikace a to opět do XML souboru. Dále program umožňuje tvorbu krátkých poznámek bez nutnosti jejich svázání s časem.

Aplikace také poskytuje možnost souběžného prohlížení více plánů od různých uživatelů na základě přístupových práv přidělených uživatelům členstvím v jednotlivých skupinách. Tímto způsobem lze sdílet jednotlivé plány v rámci skupin.

V aplikaci je zabudována podpora pro úpravu barevného nastavení aplikace s cílem poskytnout uživateli možnost upravit si aplikaci dle jeho potřeb. Nastavení aplikace je uloženo v XML souboru. Pro XML soubory vytvářené aplikací, pro export plánů, import plánů a nastavení aplikace, bylo vytvořeno XSD schéma, které slouží k validaci těchto souborů při práci s nimi.

Serverová aplikace zajišťuje zpracování požadavků od jednotlivých klientů. Zároveň také zajišťuje přístup do databáze. Součástí požadavků na aplikaci bylo i zajištění bezpečnosti v rámci komunikace mezi klientem a serverem. Tohoto požadavku bylo dosaženo pomocí symetrické šifry Blowfish. V rámci bakalářské práce také vznikl návrh na strukturu dat v databázi v jazyku SQL. Tento návrh byl posléze použit v databázi pro uložení dat jednotlivých uživatelů. Databáze byla implementována na databázovém serveru MySQL 5.0.

Jedním z hlavních požadavků na aplikaci byla multiplatformnost. Tohoto požadavku bylo dosaženo díky vhodnému výběru nástrojů pro realizaci celého projektu. Multiplatformnosti aplikace bylo dosaženo díky použití programovacího jazyka Java. K nezávislosti na OS dále napomohl databázový server MySQL. V případě konfiguračních souborů a souborů pro manipulaci s daty bylo tohoto požadavku dosaženo díky jazyku XML. Součástí bakalářské práce je také uživatelská příručka pro aplikaci MUSC.

Zkušenosti získané během vývoje této aplikace by se zajisté daly využít pro tvorbu obdobných programů, které by byly určeny pro nasazení v situaci s velkým počtem uživatelů. Určitě velmi cenné jsou zkušenosti nabyté v rámci tvorby GUI. Dále lze velmi kladně hodnotit zkušenosti získané při implementaci návrhového vzoru MVC, kde byla snaha oddělit datový model od GUI a provázat je pomocí kontroleru. Přínosná byla též nutnost manipulaci s velkými objemy dat a jejich následném efektivním a rychlým zobrazení. Ze serverové části je velmi dobře využitelná jistě část obsluhy a předávání požadavků klienta na server.

# 10 Seznam literatury


[1] Sun Microsystems, Inc.: *Java Tutorial*, www.javasoft.com

[2] Sun Microsystems, Inc.: *Java Dokumentace 1.5*, www.javasoft.com

[3] Spell Brett: *Java – Programujeme profesionálně*, Computer Press, Praha 2002

[4] Herout Pavel: *Učebnice jazyka Java*, Kopp, České Budějovice 2003

[5] Herout Pavel: *Java – grafické uživatelské prostředí a čeština,* Kopp, České Budějovice 2001

[6] Herout Pavel: *Java – bohatství knihoven*, Kopp, České Budějovice 2003

[7] Florence Tiu Balagtas: *JEDI Introduction to Programming I*, http://www.netbeans.org/community/releases/40/relnotes.html

[8] Grouchnikov Kirill: *Substance Look And Feel*, substance.dev.java.net/servlets/ProjectDocumentList

[9] Cvrček Pavel: *Začínáme s MySQL*, www.zive.cz

[10] *MySQL Dokumentace 5.0*, www.mysql.com

[11] *HTML Dokumentace 4.1*, www.w3c.org

[12] Drmola Robert, *Popis šifry Blowfish*, http://bobhy.wz.cz/

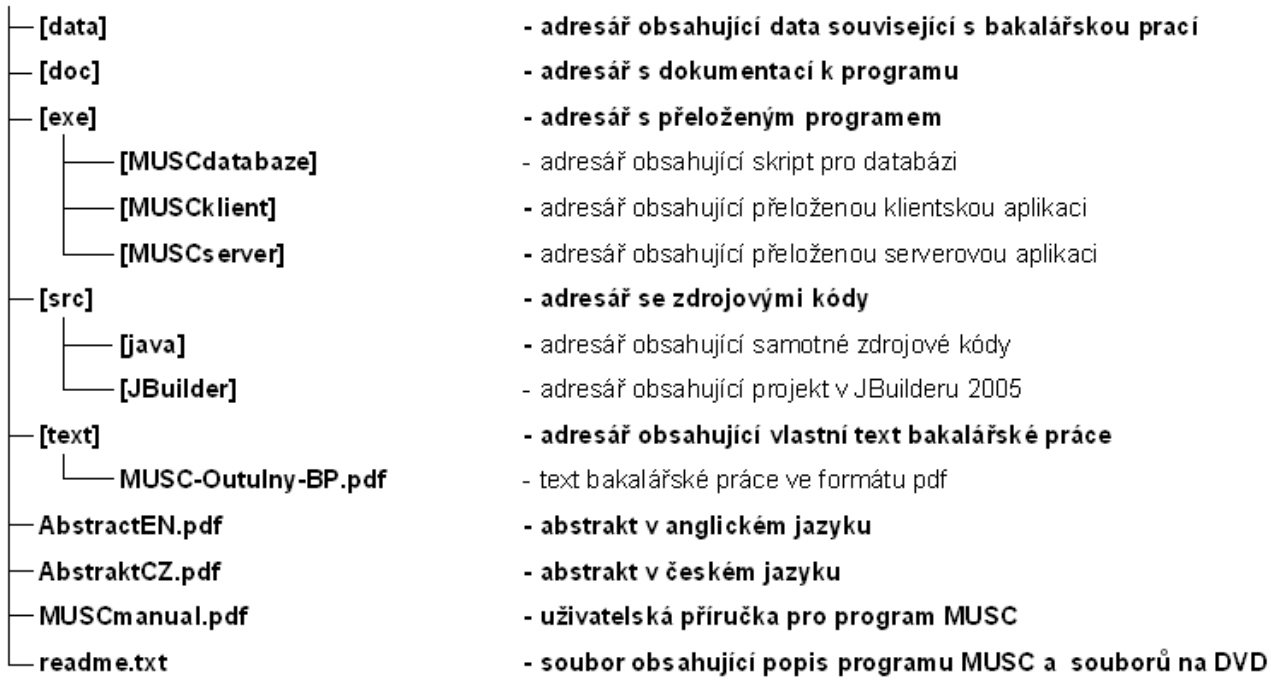
[13] *Informace o XML*, http://en.wikipedia.org/wiki/Xml

[14] Kosek Jiří: *XML pro každého – podrobný průvodce*, Grada Publishing, spol. s r. o., Praha 2000

# A Seznam použitých zkratek

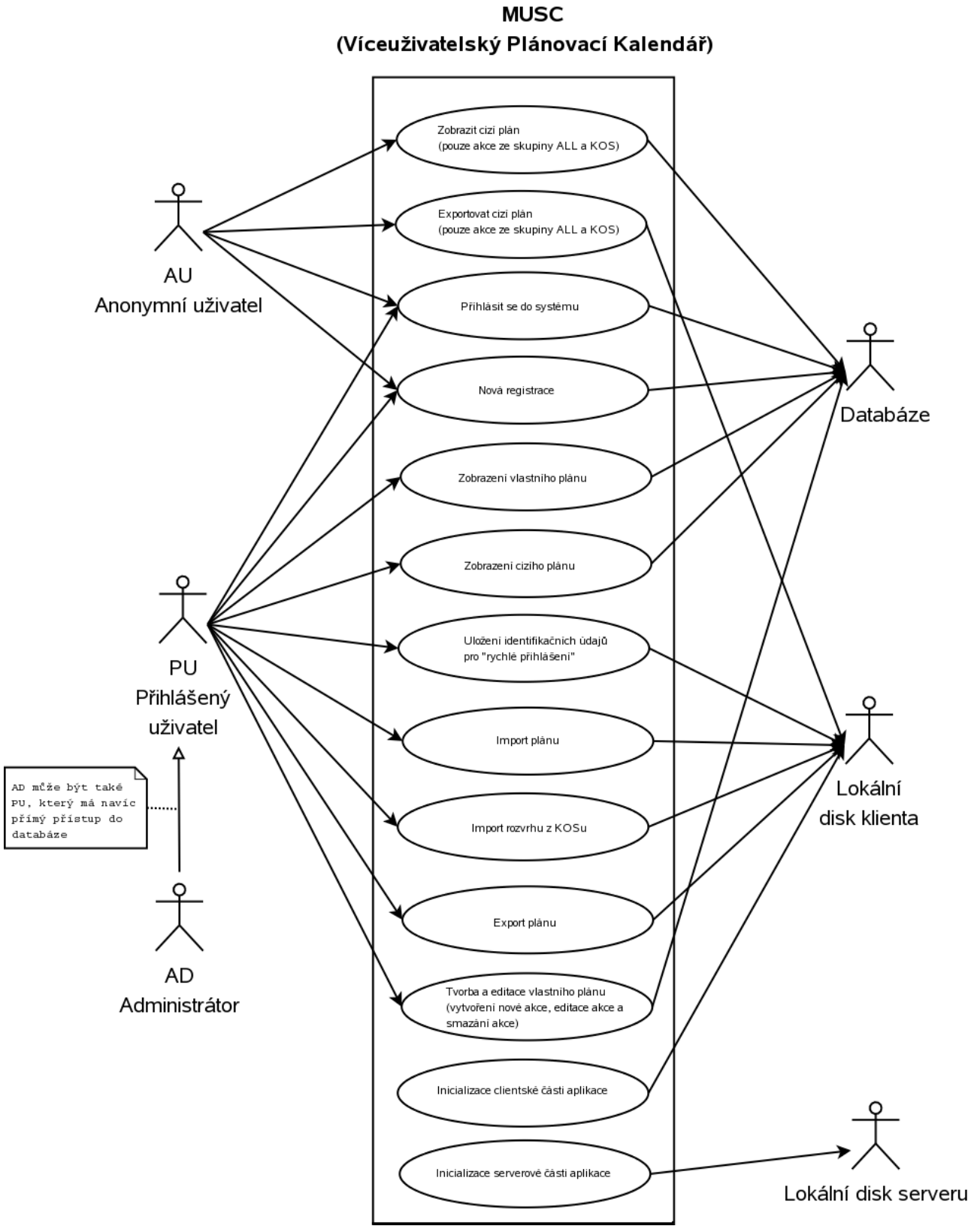
| | |
|---|---|
| **AD** | administrátor |
| **API** | Application programming interface |
| **AU** | anonymní uživatel |
| **CAST** | Carlisle Adams a Stafford Tavares (autoři šifry CAST) |
| **ČVUT** | České vysoké učení technické v Praze |
| **DES** | Data Encryption Standard |
| **DOM** | Document Object Model |
| **DTD** | Document Type Definition |
| **EBNF** | Extended BackusNaur form |
| **ER** | Entity relationship model |
| **Gtk+** | Gnu ToolKit |
| **GUI** | Graphical User Interface |
| **HP** | Hewlett Packard |
| **HTML** | HyperText Markup Language |
| **IBM** | International Business Machines Corporation |
| **JDBC** | Java Database Connectivity |
| **JRE** | Java Runtime Environment |
| **KOS** | Komponenta studia |
| **Mac OS** | Macintosh Operating System |
| **MS** | Microsoft |
| **MS SQL** | Microsoft SQL Server |
| **MUSC** | Multiuser Scheduling Calendar = Víceuživatelský Plánovací Kalendář |
| **MVC** | Model view controller |
| **MySQL** | My SQL |
| **OS** | operační systém |
| **Pro** | professional |
| **PU** | přihlášený uživatel |
| **SAX** | Simple API for XML |
| **SGML** | Standard Generalized Markup Language |
| **SP2** | Service pack 2 |
| **SQL** | Structured Query Language |
| **UML** | Unified Modeling Language |
| **W3C** | World Wide Web Consortium |
| **XML** | Extensible Markup Language |
| **XP** | experience |
| **XSD** | XML Schema Definition |

# B Obsah přiloženého DVD

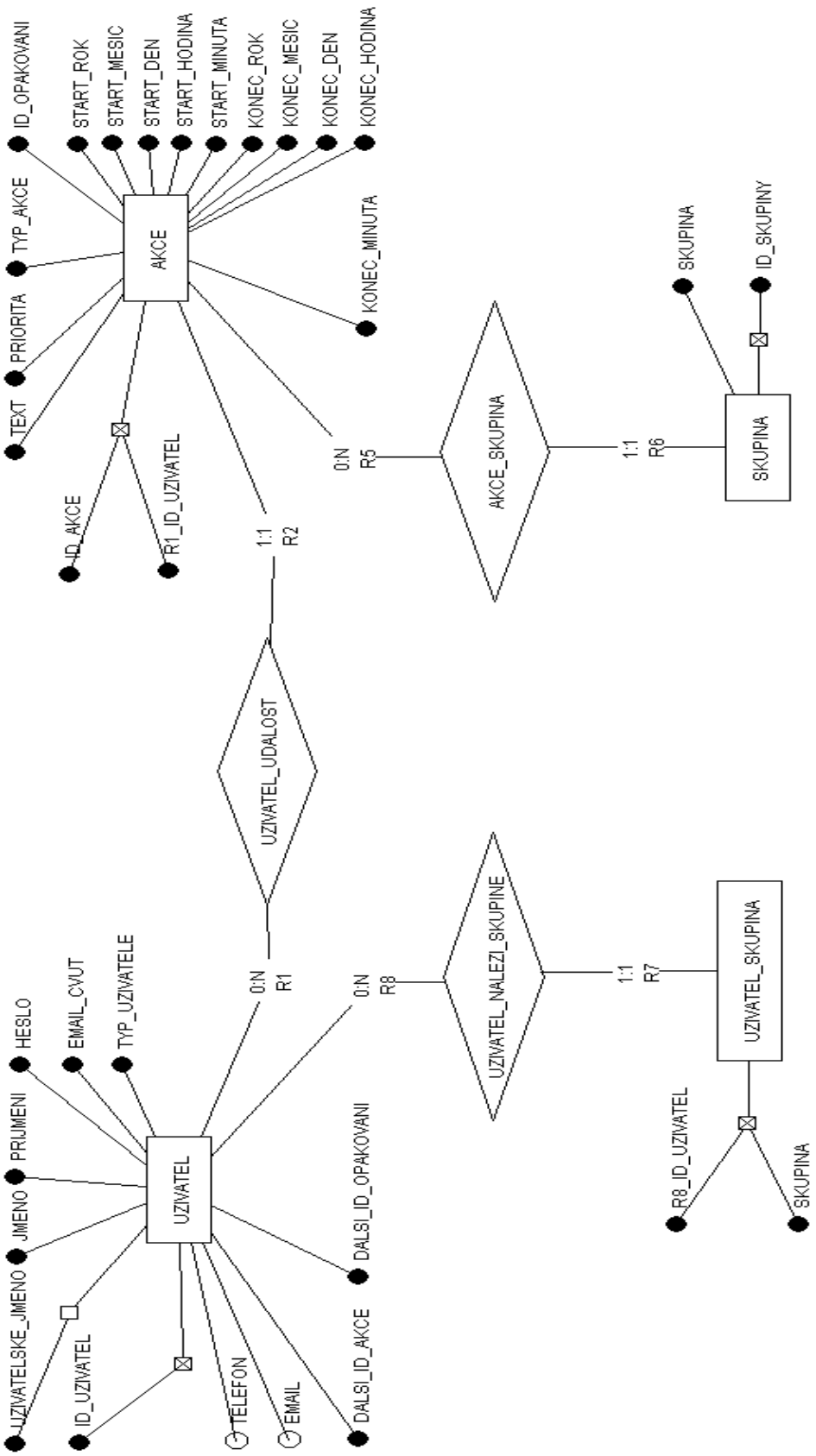


# C Seznam příloh

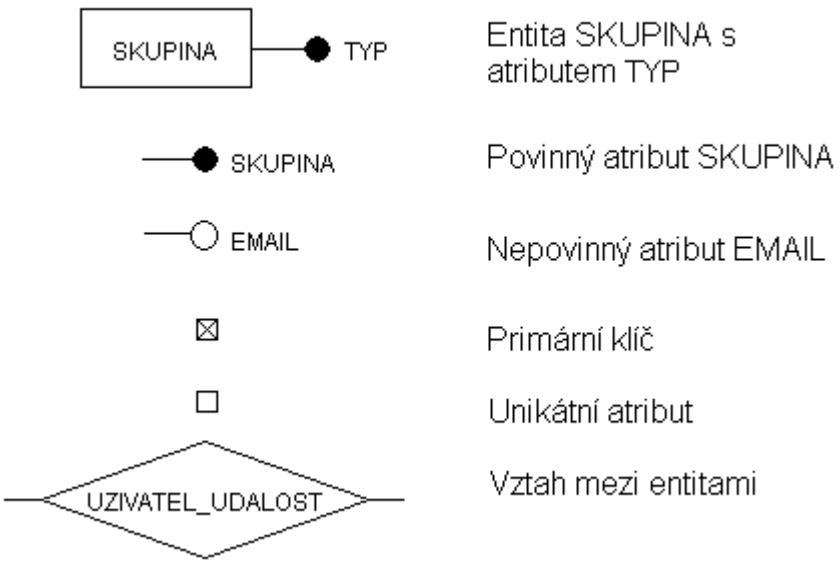
1. UML diagram – model jednání.
2. ER model dat v databázi.
3. ER diagram – anotace.




**Příloha 1: UML diagram – model jednání**




**Příloha 2: ER model dat v databázi**




**Příloha 3: ER diagram – anotace**