VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Robustní reimplementace prototypového řešení serverových komponent systému Virtlab

Robust Reimplementation of Prototype Virtlab System Server Components

2010 Radek Novák

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2010 .....

Rád bych na tomto místě poděkoval mému vedoucímu Petrovi Grygárkovi za cenné přípomínky a nápady, dále pak mým kolegům Kateřině Bambuškové a Martinu Milatovi za pomoc při testování a řešení technických problémů. Všem výše jmenovaným děkuju za dva roky přínosné a příjmené spolupráce na projektu Virtlab.

Abstrakt

Tato diplomová práce se zabývá reiplementací serverových komponent systému Virtlab. Současná prototypová implementace není dostatečně robustní, není vytvořena s pomocí objektově orientovaných technologií a postrádá možnost snadné rozšířitelnosti. Nová implementace bude vytvořena v souladu se zásadami Softwarového inženýrství, bude dostatečně robustní, využívat objektově orientované technologie a bude obsahovat rozšířenou funkcionalitu, která vyplynula z dvouletého využívání systému Virtlab.

Klíčová slova: Virtlab, Softwarové inženýrství, Objektově orientované technologie, Distribuované technologie

Abstract

This diploma thesis dealt with reimplementation of system Virtlab server's components. Current prototype implementation is not enought robust, it is not created with object oriented technologies and it´s lack ability of easy expandibility. New implementation will be implemented under principles of Software enginiering, it will be robust, it will be implimented with usegae of object oriented technologies and it will contain extended functionality which emerges from two year usage of system Virtlab.

Keywords: Virtlab, Software enginiering, Object oriented technologies, Distributed technologies

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
HTTP	–	Hyper Text Transfer Protocol
WWW	–	World Wide Web
PHP	–	Professional Home Pages
UML	–	Unified Modeling Language
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
IP	–	Internet Protocol
XML	–	Extensible Markup Language
MVC	–	Model View Controller
DAO	–	Database Access Object
ORM	–	Object Relations Mapping
FD	–	File Descriptor
ID	–	IDentificator
Q-in-Q	–	Tunelování 802.1q rámců v 802.1q rámcích
VLAN	–	Virtual Local Area Network
BASH	–	Bourne-Again Shell
SSL	–	Secure Socket Layer
VT100	–	Video Terminal 100
OS	–	Operation System

Obsah

Seznam obrázků

1 Úvod

Distribuovaná Virtuální laboratoř počítačových sítí umožňuje zpřístupnění reálných síťových prvků, kterými laboratoř disponuje, uživatelům skrze internet. Uživatelé si rezervují úlohy na určitý čas. V daném čase pak s úlohami pracují, nastavují síťové prvky, testují je a pod. V posledních letech roste počet uživatelů virtuální laboratoře. Je to dáno mimo jiné rozšířením počtu lokalit distribuované laboratoře a mimo jiné vzrůstajícím zájmem o počítačové sítě. Díky nárustu počtu uživatelů vyvstanuly nové problémy, které je nutno řešit. Jsou to:

- **Možnost pracovat na zařízení ve více lidech současně** - tento požadavek vyvstává z reálné situcae, která je k vidění při praktické výuce počítačových sítí. Častokrát se stává, že skupina pracuje společně na jednom zařízení a řeší problémy, které se daného zařízení týkají.
- **Získat přístup na zařízení v různých módech** - tento požadavek vzniknul hlavně díky nasazení nového automatizovaného testovacího systému, který potřebuje získat exkluzivní přístup k danému zařízení, spustit na něm testovací skripty a pak toto zařízení vrátit studentům.
- **Důkladná analýza a logování příkazů zapsaných na zařízení** - velice užitečný požadavek, který umožňuje kontrolovat, co uživatel zapsal na zařízení a zrekonstruovat zpětně jeho celou práci se zařízením.
- **Nasadit mezi uživatelská data a zařízení různé filtry** - další z velice užitečných požadavků, který umožňuje upravovat uživatelská data. Ať už se jedná o logování, zakazování určitých příkazů anebo o kontrolu toku dat, tzv. flow controll.
- **Možnost zjistit připojené uživatele na zařízení a popřípadě je odpojit** - tento požadavek vyšel z potřeby administrátorů zjišťovat a popřípadě odpojovat uživatele připojené k zařízením.
- **Definovat rozvrhy pro zařízení** - tento požadavek vznikl na základě nutnosti omezit v určitých hodinách práci uživatelů s Virtuální laboratoří a definovat určitá časová okna určená pro údržbu systému. Taktéž je nutno tyto rozvrhy aplikovat pouze na určité uživatele anebo skupiny uživatelů.
- **Definovat odstávky zařízení** - tento požadavek vznikl na základě nutnosti odstavit určitá zařízení a neposkytovat je pro účely virtuálních rezervací. Pokud má například některé ze zařízení poruchu, pak je nutné jej dočasně odstavit.

Tato diplomová práce si klade za cíl reimplementovat serverovou část virutální laboratoře respektive Rezervační a Konzolový server. Současná implementace není vhodná pro implementaci nových požadavků a nerespektuje zásady a metody softwarového inženýrství. Proto bude vyvořena nová implementace zahrnující veškerou současnou funkčnost spolu s novými požadavky a bude implementována s použitím moderních metod softwarového inženýrství.

2 Architektura distribuované Virtuální laboratoře

2.1 Distribuovaná Virtuální laboratoř(Virtlab)

Vznik projektu Virtuální laboratoře byl iniciován v roce 2005 v diplomové práci Pavla Němce. Jednalo se však pouze o nedistribuovanou varanatu, která neumožňovala dynamické spojování topologií. To se umožnilo až díky diplomové práci Davida Saidela, který navrl zařízení ASSSK pro automatické spojování rezervací. Zabezpečení Virtuální laboratoře pak bylo dopracováno v diplomové práci Romana Kubína.

Jelikož je každá laboratoř tvořena nákladnými zařízeními a tato zařízení nejsou vždy plně využita, byla přijata myšlenka vytvořit Virtuální laboratoř distribuovaně. Distribuovanost obnáší existenci více lokalit Virtuálních laboratoří, které si vzájemně poskytují zařízení vhodná do Virtuálních topologií. Architektura distribuované Virtuální laboratoře byla popsána v diplomových pracích Jana Vavříčka a Tomáše Hrabálka. Taktéž bylo v tomto roce vyvinuto modernizované zařízení pro automatické spojování konfigurací nazvané ASSSK2. Vývoj tohoto zařízení je popsán v diplomové práci Petra Sedláře.

Jako další přínosy pro Virtuální laboratoř můžeme uvést například implementaci sledování provozu na VLAN popsanou v bakalářské práci autora této diplomové práce, modernizaci jedné z klíčových komponent Virtuální laboratoře - Tunelovacího serveru popsané v bakalářské práci Václava Bortlíka nebo implementaci automatických testů popsanou v diplomové práci Zdeňka Filipce. [1]

2.2 Architektura distribuované Virtuální laboratoře

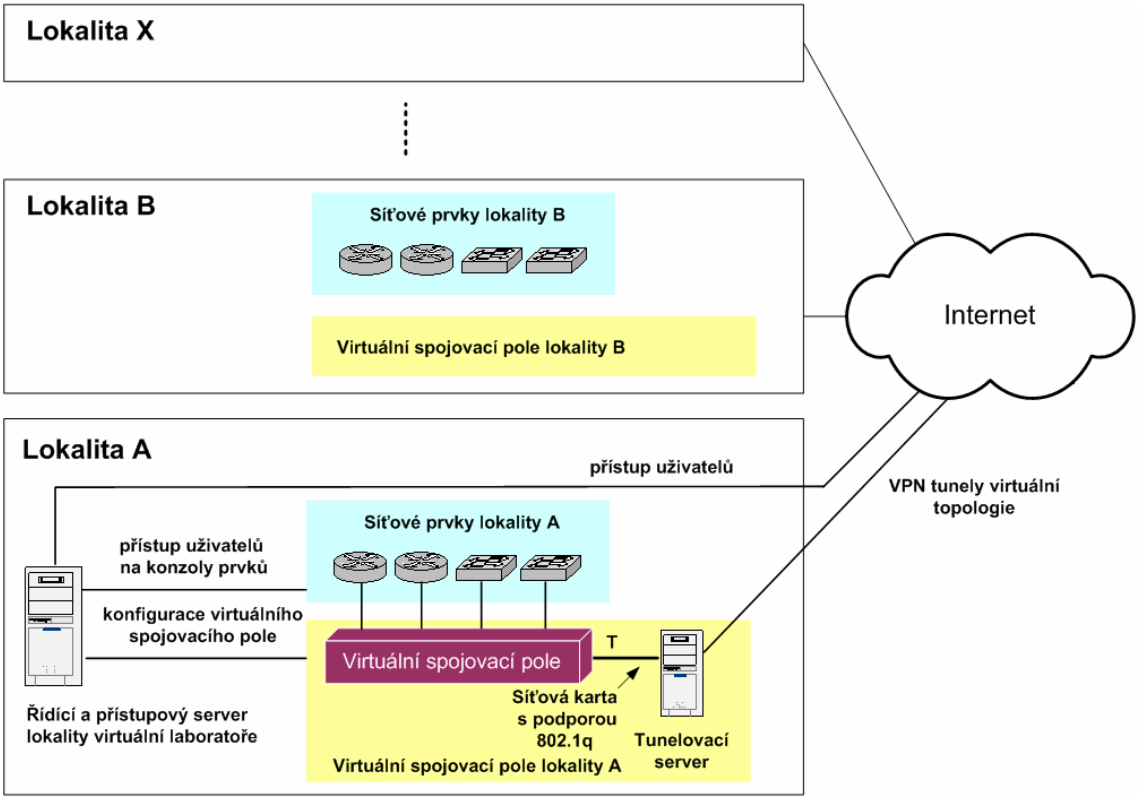
Celá distribuovaná virtuální laboratoř se skládá z tzv. lokalit, viz. obrázek 1. Lokalitou se rozumí soubor softwarových a jiných technických prostředků, které tvoří samostatnou funkční jednotku virtuální laboratoře. Tato funkční jednotka samozřejmě může komunikovat s dalšími lokalitami a využívat jejich zařízení. Každá lokalita je tvořena třemi důležitými částmi.

1. **Řídící webová aplikace** - Jedná se o webovou aplikaci, se kterou komunikují uživatelé Virtuální laboratoře a skrze kterou si rezervují své úlohy, přistupují na zařízení, píšou si mezi sebou e-maily a pod.

2. **Serverová část** - Jedná se o mezivrstu mezi webovou aplikací a zařízeními. Zajišťuje přeposílání datových toků mezi jednotlivými lokalitami, umožňuje přístupy na zařízení a taktéž spravuje a půjčuje zařízení pro rezervace.

3. **Zařízení** - Jedná se o reálné síťové prvky anebo zařízení nutné pro běh virtuální síťové laboratoře. Jako třeba zařízení ASSSK nebo MOXA karty.

[1] Veškerý soupis diplomových prací lze nalézt na oficiálních stránkách projektu [5], více informací o terminologii systému Virtlab lze nalézt na [4]



Obrázek 1: Architektura distribuované virtuální laboratoře

2.3 Obecné pojmy definující distribuovanou virtuální laboratoř

1. **Lokalita** - Lokalitou rozumíme jednu samostatnou virtuální laboratoř v distribuované struktuře se vším softwarovým a hardwarovým zařízením, schopnou samostatné práce nezávisle na ostatních lokalitách.

2. **Uživatelská skupina** - Uživatelskou skupinou rozumíme jednu ze skupin, do které jsou organizováni uživatelé virtuální laboratoře. Jedná se např. o skupinu: studenti, užitelé, administrátoři a pod.

3. **Cizí lokalita** - Cizí lokalitou rozumíme jednu samostatnou virtuální laboratoř v distribuované struktuře se vším softwarovým a hardwarovým zařízením, schopnou samostatné práce nezávisle na ostatních lokalitách.

2.4 Architektura serverové vrstvy

Tato diplová práce se týká druhé vrstvy. Proto se na ní podíváme důkladněji. Serverová vrstva virtuální laboratoře se skládá ze tří důležitých softwarových komponent. Jsou jimi:

- **Tunelovací server** - Stará se o přeposílání datových toků mezi jednotlivými lokalitami Virtuální laboratoře. Poprvé byl implementován v diplomové práci Tomáše Hrabálka a později reimplementován v bakalářské práci Václava Bortlíka. Do bezproblémového provozu byl pak později uveden autorem této diplomové práce. Tunelovací server pracuje jednoduše. Zachytává veškerý provoz na určeném rozhraní serveru lokality, na kterou je sveden veškerý provoz mezi zařízeními, které se nacházejí ve virtuální laboratoři. Pokud tento provoz nenáleží do této lokality, pak je zabalen ve formátu UDP rámce a poslán skrze internet do lokality, ke které náleží. Zde je rozbalen a skrze rozhraní poslán určenému zařízení. Identifikace provozu pro určené zařízení je pak realizována pomocí technologie Q-in-Q, která je realizována nad technoloigí VLAN.

- **Konzolový server** - Byl poprvé implementován již v nedistribuované verzi virtuální laboratoře Pavlem Němcem. Později byl rozšířen o práci v distribuovaném prostředí Tomášem Hrabálkem. Konzolový server se stará o zpřístupnění konzole daného síťového prvku uživateli. Uživatel komunikuje s Konzolovým serverem pomocí Java appletu, který simuluje chování konzole daného zařízení. Konzolový server pak komunikuje s daným zařízením skrze sériovou linku, TCP spojení anebo slouží pouze jako proxy mezi uživatelem a Konzolovým serverem cizí lokality, do které zařízení náleží. Taktéž se stará o verifikaci uživatele a kontroluje zakázané příkazy.

- **Rezervační server** - Byl poprvé implementován v rámci diplomové práce Tomáše Hrabálka. Jedná se o správce zařízení, které má na starosti uchovávat informace o zarezervovaných zařízeních, poskytovat seznam zařízení, rušit rezervace na zařízení a pod. Veškeré tyto požadavky samozřejmě plní v distribuované tak nedistribuované podobě. Ke korektnímu rezervování zařízení v distribuovaném prostředí využívá speciální navržený dvoufázový potvrzovací protokol. Další důležitou úlohou Rezervačního serveru je naplánování aktivačních a deaktivačních skriptů, které se starají o uvedení dané rezervace do chodu.

V této diplomové práci se budeme soustředit na reimplementaci Rezervačního a Konzolového serveru.

3 Rezervační server

Rezervační server slouží jako správce zařízení, která mohou být využívána pro potřeby virtuálních topologií. Stará se o poskytování seznamu zařízení, tvorbu rezervací, rušení rezervací a v neposlední řadě taktéž o aktivaci rezervací. Veškeré tyto požadavky plní samozřejmě jak v distribuované tak nedistribuované podobě.

3.1 Terminologie Rezervačního serveru

U Rezervačního serveru budeme používat následující terminologii:

- **Rezervace** - Rezervací rouzmíme množinu zařízení, která budou použity ve virtuální topologii. Rezervace má daný čas konce, začátku a ID uživatele kterému patří.
- **Potvrzení** - Potvrzením rozumíme požadavek na Rezervační server, který oznamuje, že rezervace v distribuované formě proběhla bez problému a může být zaznamenána do databáze.
- **Odstávka** - Odstávkou rozumíme množinu zařízení, která nebudóu v daném časovém období poskytovány pro potřeby virtuálních topologií.
- **Časovým rozvrh** - Časovým rozvrhem rozumíme množinu zařízení, která můžou být v daném časovém období a daných časových oknech použita pro potřeby virtuálních topologií.
- **Omezující podmínka** - Omezující podmínkou rozumíme podmínku, která musí být splěna aby mohla být daná množina zařízení použita pro potřeby virtuálních topologií. Např. zařízení nesmí být v daném období použito v jiné rezervaci a pod.
- **Požadavek** - Požadavkem rozumíme příkaz, který je zaslán pomocí definovaného Komunikačního protokolu Rezervačnímu serveru. Např. příkaz pro vytvoření rezervace a pod.
- **Aktivační skript** - Skript v jazyce BASH, který slouží ke spuštění rezervace.
- **Deaktivační skript** - Skript v jazyce BASH, který slouží k ukončení rezervace.

4 Rezervační server - Specifikace požadavků

4.1 Definice jednotlivých aktérů

S Rezervačním serverem pracují tři druhy aktérů. Každý z nich má určité očekávání a cíle, které potřebuje splnit, viz. obrázek 2. Aktéři jsou tedy následující [2]:

Řídící aplikace

Jako aktéra Řídící aplikaci bereme webovou aplikaci, se kterou přímo komunikuje uživatel Virtlabu. Řídící aplikace(dále jen Aplikace) nabízí uživateli možnost vytvořit si rezervaci v určitém časovém intervalu. Taktéž umožňuje uživateli vidět již vytvořené rezervace a zjistit tím, ve kterém časovém období bude dostatek zařízení pro jeho rezervaci.

S Rezervačním serverem komunikuje pouze část Aplikace, která se stará o rezervaci virtuálních topologií. Komunikační rozhraní mezi Aplikací a Rezervačním serverem je pevně definované, odolné vůči chybám a uživatel nemá přímou možnost, jak ovlivnit komunikaci nebo data, která jsou přenášena mezi Aplikací a Rezervačním serverem.

Rezervační server jiné lokality

Jako aktéra Rezervační server jiné lokality (dále jen Jiný server) bereme aplikaci Rezervačního serveru, která se nachází v jiné lokalitě Virtuální laboratoře. Jiný server komunikuje s našim Rezervačním serverem, pokud zjišťuje dostupná zařízení pro distribuovanou virtuální topologii anebo pokud se pokouší tuto distribuovanou virtuální topologii zarezervovat, popřípadě jí zrušit. Je nutno poznamenat, že tato interakce proběhne pouze tehdy, když se jedná o distribuovanou virtuální topologii a když má Jiný server uvedenou ve svém konfiguračním souboru naši lokalitu Virtuální laboratoře.

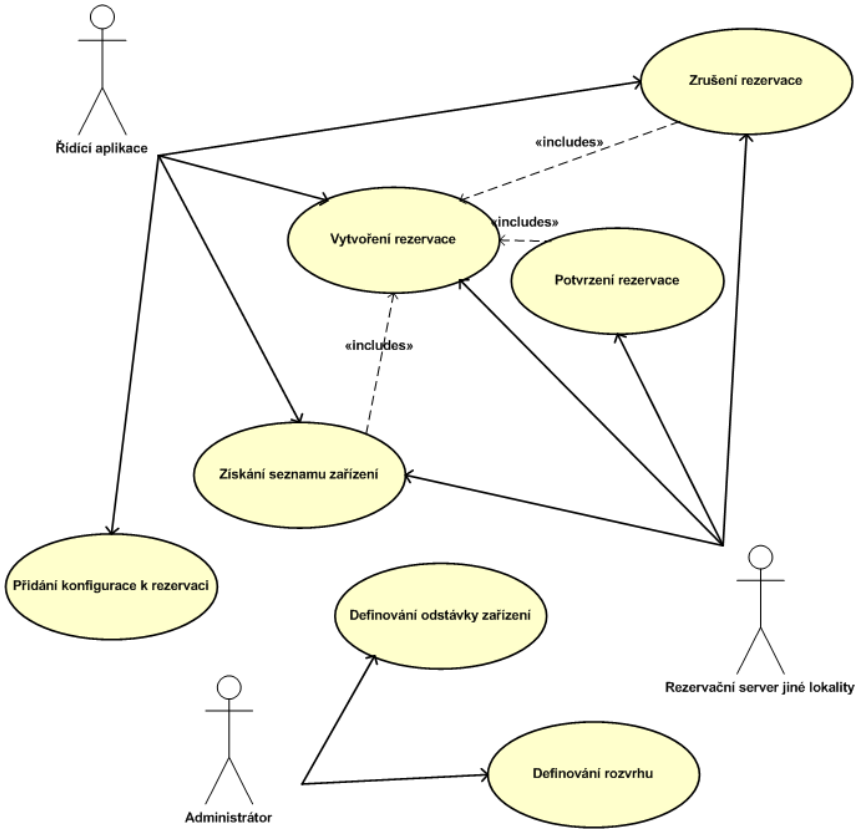
Mezi rezervačními servery probíhá stejná komunikace jako mezi Řídící aplikací a Rezervačním serverem. Rozhraní je opět pevně dané a uživatel Virtuální laboratoře nemá možnost změnit podobu dat, která jsou mezi servery přenášena.

Administrátor

Jako aktéra Administrátora bereme správce lokality Virtuální laboratoře, který má přístup k serverovým komponentám Virtlabu a jejich řídících rozhraních. Administrátor přistupuje k serverovým komponentám, pokud potřebuje změnit rozpis rozvrhů pro dané časové období anebo pokud potřebuje provést odstávku zařízení na určité časové období.

Administrátor komunikuje s Rezervačním serverem skrze ovládací konzoli, kterou má každá serverová komponenta v sobě obsaženou.

[2] Styl psaní případů užití je převzat z knihy [11]



Obrázek 2: USE-CASE diagram cílů jednotlivých aktérů

4.2 Případ užití č.1 - Získání seznamu zařízení

Popis případu užití

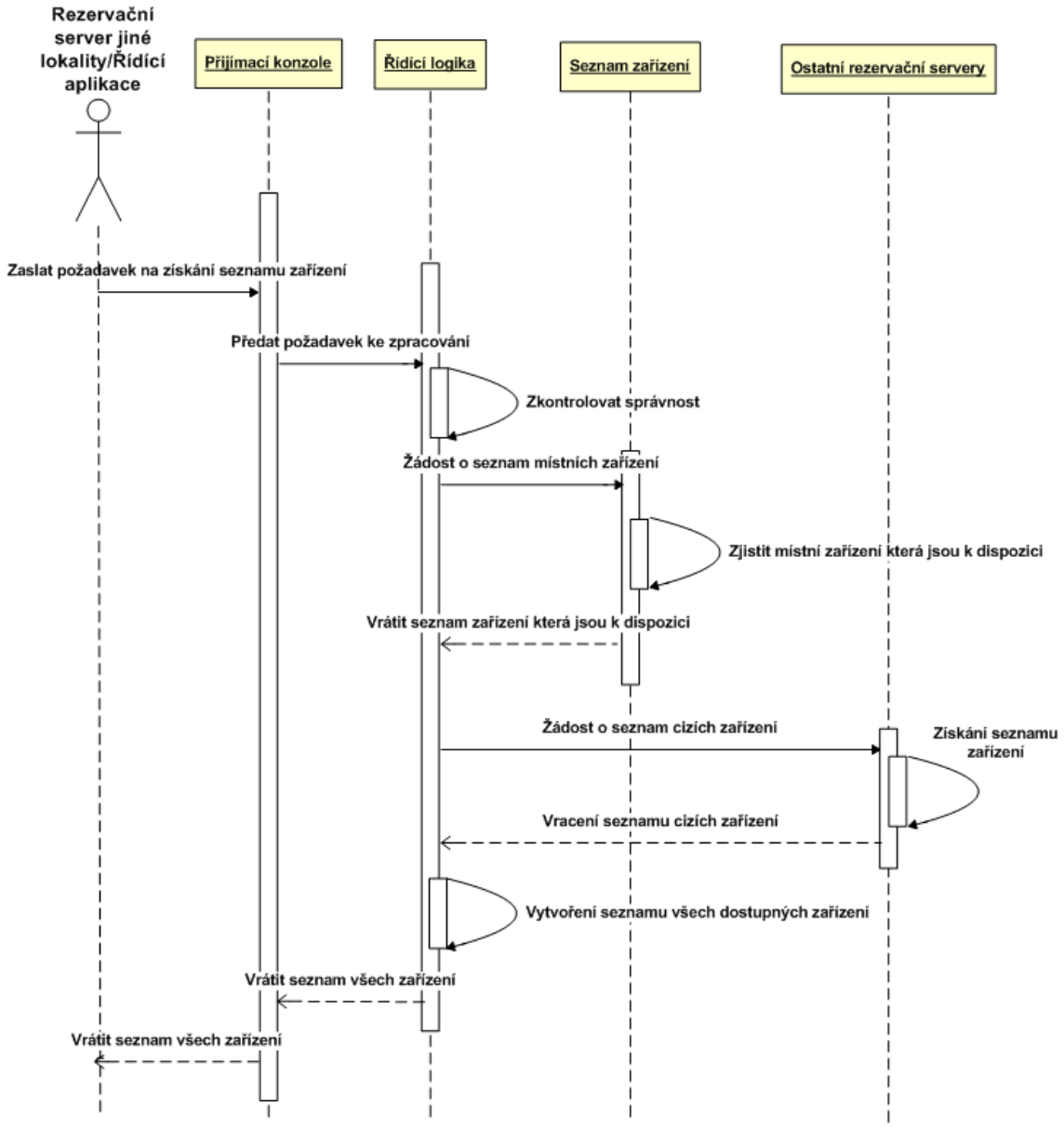
Řídící aplikace nebo jiný Rezervační server zažádá o seznam zařízení v dané lokalitě Virtuální laboratoře. Na Rezervačním serveru je, aby zjistil, která zařízení může v dané době nabídnout. Musí brát v potaz zařízení, které jsou již obsažena v jiných rezervacích, čekají na potvrzení, jsou v odstávce anebo je nedovoluje poskytnout jeden z definovaných rozvrhů. Pokud Řídící aplikace specifikuje, že se bude jednat o distribuovanou rezervaci, pak musí Rezervační server kontaktovat Rezervační servery ostatních lokalit a vyžádat si seznamy jejich dostupných zařízení. Při kontaktování ostatních lokalit již nepožaduje distribuovanou verzi příkazu. Grafické znázornění základního toku lze vidět na obrázku 3.

Primární aktéři

- Řídící aplikace
- Rezervační server jiné lokality

Základní tok událostí

1. Řídící aplikace nebo Rezervační server jiné lokality zažádají o seznam zařízení.
2. Rezervační server přijme požadavek pomocí definovaného Komunikačního protokolu a vytvoří nový požadavek.
3. Rezervační server zkontroluje, zda je nový požadavek na distribuovanou rezervaci nebo není.
4. Pokud je požadavek distribuovaný, pak Rezervační server kontaktuje Rezervační servery jiných lokalit pomocí daného Komunikačního protokolu.
5. Pokud je požadavek distribuovaný, pak Rezervační servery jiných lokalit jako odpověď pošlou seznam zařízení, která v daném časovém období mohou poskytnout.
6. Rezervační server zjistí, která zařízení může poskytnout ze své lokality.
7. Rezervační server zjistí, zda nejsou v daném časovém období již některá lokální zařízení rezervována.
8. Rezervační server zjistí, zda v daném časovém období již některá lokální zařízení nečekají na potvrzení rezervace.
9. Rezervační server zjistí, zda nejsou lokální zařízení v daném časovém období v odstávce.
10. Rezervační server zjistí, zda v daném časovém období neomezuje poskytování lokálních zařízení časový rozvrh.
11. Rezervační server poskytne všechna možná lokální zařízení.
12. Rezervační server zesumarizuje seznam zařízení od jiných Rezervačních serverů spolu se seznam lokálních zařízení, která může poskytnout.
13. Rezervační server vrátí seznam zařízení pomocí definovaného Komunikačního protokolu Řídící aplikaci nebo Rezervačnímu serveru jiné lokality.



Obrázek 3: Sekvenční diagram základního toku Získání seznamu zařízení

4.3 Případ užití č.2 - Vytvoření rezervace

Popis případu užití

Řídící aplikace nebo jiný Rezervační server zažádá o vytvoření rezervace v dané lokalitě. Na Rezervačním serveru je, aby zjistil, zda jsou zařízení, ze kterých se bude rezervace skládat v daném časovém období, k dispozici. Postupuje stejně jako v Případě užití č.1. Pokud je přímo specifikováno, že se bude jednat o distribuovanou rezervaci, pak musí Rezervační server zjistit, ze kterých lokalit zařízení pocházejí a zažádat o jejich zarezervování v jiných lokalitách. Při kontaktování ostatních lokalit již nepožaduje distribuovanou verzi příkazu. Grafické znázornění základního toku lze vidět na obrázku 4.

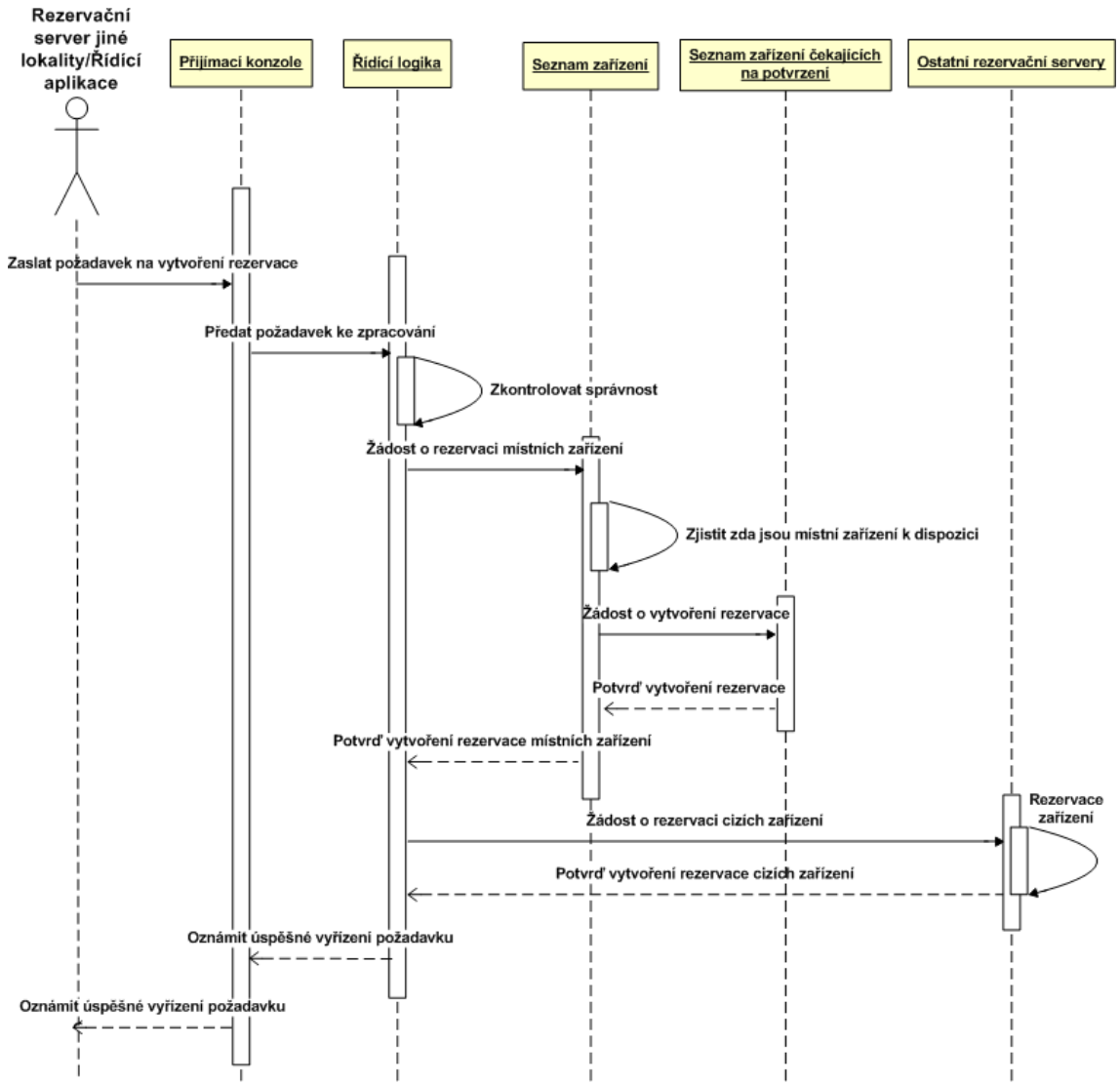
Primární aktéři

- Řídící aplikace
- Rezervační server jiné lokality

Základní tok událostí

1. Řídící aplikace nebo Rezervační server jiné lokality zažádají o vytvoření rezervace.
2. Rezervační server přijme požadavek pomocí definovaného Komunikačního protokolu a vytvoří nový požadavek.
3. Rezervační server zkontroluje, zda je nový požadavek na distribuovanou rezervaci nebo není.
4. Rezervační server zjistí, která zařízení v rezervaci se vztahují na tuto lokalitu.
5. Rezervační server zjistí, zda nejsou v daném časovém období již některá lokální zařízení rezervována.
6. Rezervační server zjistí, zda v daném časovém období již některá lokální zařízení nečekají na potvrzení rezervace.
7. Rezervační server zjistí, zda nejsou lokální zařízení v daném časovém období v odstávce.
8. Rezervační server zjistí, zda v daném časovém období neomezuje poskytování lokálních zařízení časový rozvrh.
9. Rezervační server zjistí, zda jsou lokální zařízení patřící do rezervace obsažena v seznamu zařízení, která může pro dané časové období nabídnout.
10. Rezervační server vytvoří rezervaci na lokální zařízení a nechá ji čekat na potvrzení.

11. Rezervační server kontaktuje rezervační servery jiných lokalit pomocí daného Komunikačního protokolu a zašle jim žádosti o rezervace zařízení v příslušných lokalitách.
12. Rezervační servery jiných lokalit jako odpověď pošlou infromaci, zda se rezervace povedla nebo ne.
13. Rezervační server zjistí, zda se rezervace povedly ve všech lokalitách, kterých se rezervace týkaly.
14. Rezervační server potvrdí všechny rezervace.
15. Rezervační server zjistí, zda se povedla všechna potvrzení.
16. Rezervační server si poznamená infromaci o existující rezervaci.
17. Rezervační server vrátí odpověď o úspěšné rezervaci pomocí definovaného Komunikačního protokolu Řídící aplikaci nebo Rezervačnímu serveru jiné lokality.



Obrázek 4: Sekvenční diagram základního toku Vytvoření rezervace

4.4 Případ užití č.3 - Zrušení rezervace

Popis případu užití

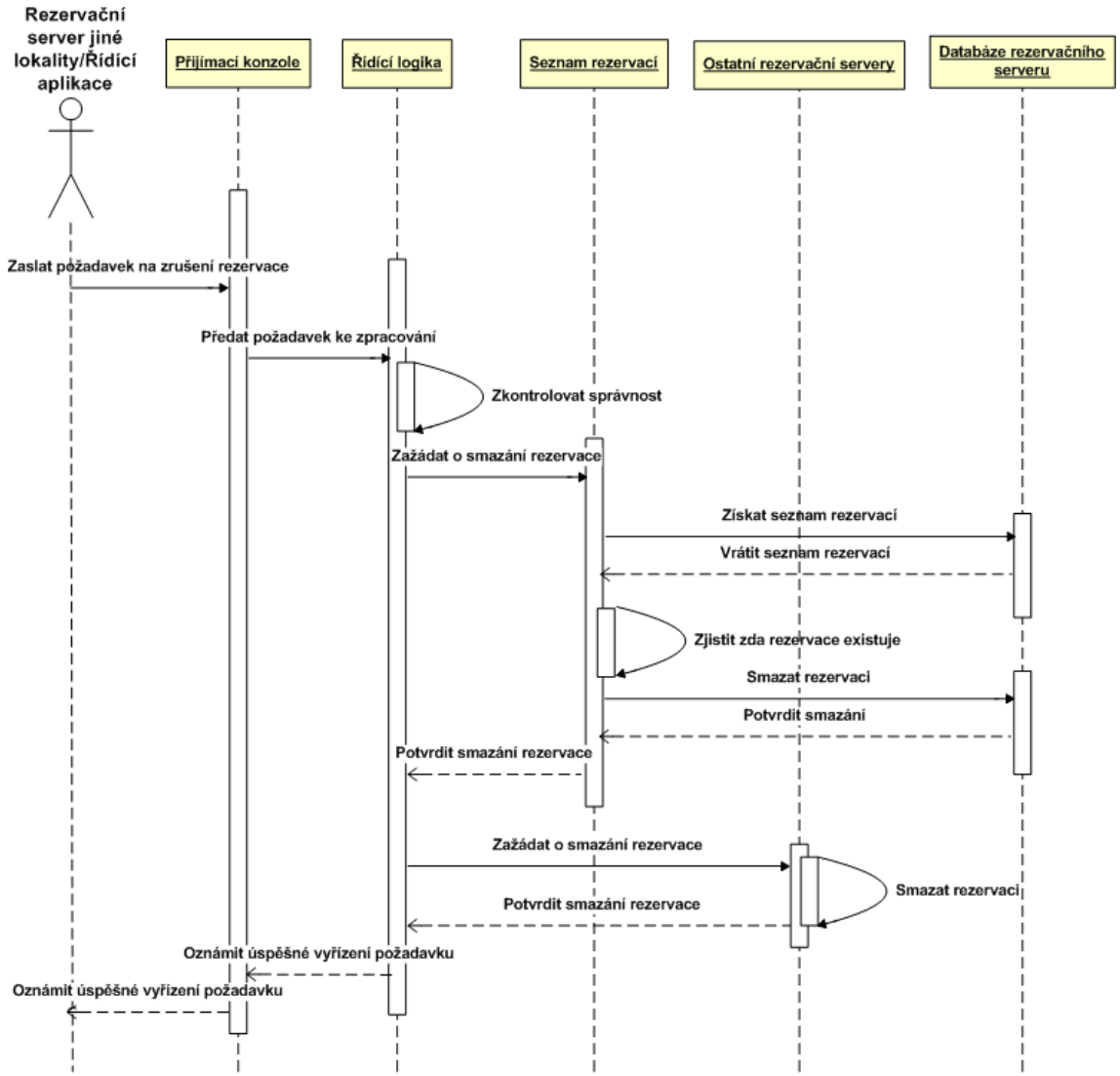
Řídící aplikace nebo jiný Rezervační server zažádají o zrušení rezervace v dané lokalitě. Na Rezervačním serveru je, aby zjistil, zda daná rezervace existuje, zda již nebyla spuštěna a neprobíhá anebo zda se jedná o distribuovanou rezervaci. Pokud se jedná o distribuovanou rezervaci, pak musí Rezervační server zažádat o zrušení rezervace ve všech lokalitách. Pokud již rezervace probíhá, pak není rezervace zrušena a je pouze posunut čas ukončení rezervace. Zrušení rezervace se taktéž používá pro zrušení rezervací, které ještě nebyly potvrzeny. Při kontaktování ostatních lokalit již nepožaduje distribuovanou verzi příkazu. Grafické znázornění základního toku lze vidět na obrázku 5.

Primární aktéři

- Řídící aplikace
- Rezervační server jiné lokality

Základní tok událostí

1. Řídící aplikace nebo Rezervační server jiné lokality zažádají o zrušení rezervace.
2. Rezervační server přijme požadavek pomocí definovaného Komunikačního protokolu a vytvoří nový požadavek.
3. Rezervační server zkontroluje, zda je nový požadavek na distribuovanou rezervaci nebo ne.
4. Rezervační server zjistí, zda má informaci o rezervaci, která je určená k zrušení.
5. Rezervační server zjistí, jestli již rezervace běží, doběhla nebo je naplánovaná do budoucna.
6. Rezervační server zruší rezervaci a odstraní si informaci o ní.
7. Rezervační server rozešle požadavek na zrušení rezervace ostatním Rezervačním serverům.
8. Rezervační servery jiných lokalit jako odpověď pošlou infromaci, zda se rezervace zrušila nebo ne.
9. Rezervační server vrátí odpověď o úspěšném zrušení rezervace pomocí definovaného Komunikačního protokolu Řídící aplikaci nebo Rezervačnímu serveru jiné lokality.



Obrázek 5: Sekvenční diagram základního toku Zrušení rezervace

4.5 Případ užití č.4 - Potvrzení rezervace

Popis případu užití

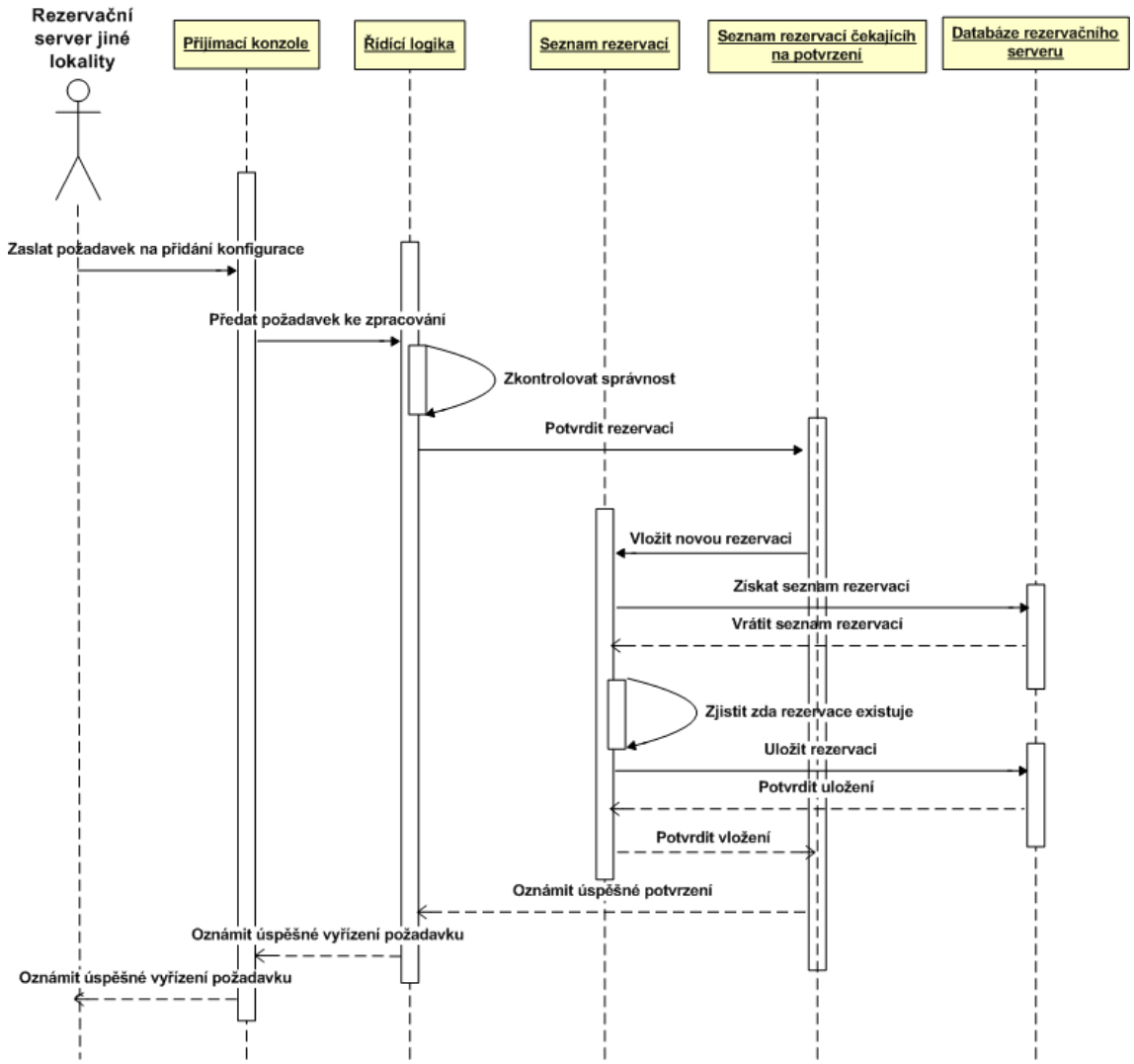
Rezervační server jiné lokality zažádá o potvrzení rezervace v dané lokalitě. Na cílovém Rezervačním serveru je, aby zjistil, zda daná rezervace existuje, čeká na potvrzení a nevypršel jí časový limit pro potvrzení. Pokud jsou všechny podmínky úspěšně splněny, pak je nová rezervace vytvořena a Rezervační server si o ní uloží informace. Grafické znázornění základního toku lze vidět na obrázku 6.

Primární aktéři

- Rezervační server jiné lokality

Základní tok událostí

1. Rezervační server jiné lokality zažádá o potvrzení rezervace.
2. Rezervační server přijme požadavek pomocí definovaného Komunikačního protokolu a vytvoří nový požadavek.
3. Rezervační server zjistí, zda má informaci o rezervaci, která je určená k potvrzení.
4. Rezervační server zjistí, jestli již nevypršel časový limit pro potvrzení.
5. Rezervační server potvrdí rezervaci a uloží si informace o ní.
6. Rezervační server vrátí odpověď o úspěšném potvrzení rezervace pomocí definovaného Komunikačního protokolu Rezervačnímu serveru jiné lokality.



Obrázek 6: Sekvenční diagram základního toku Potvrzení rezervace

4.6 Případ užití č.5 - Přidání konfigurace k rezervaci

Popis případu užití

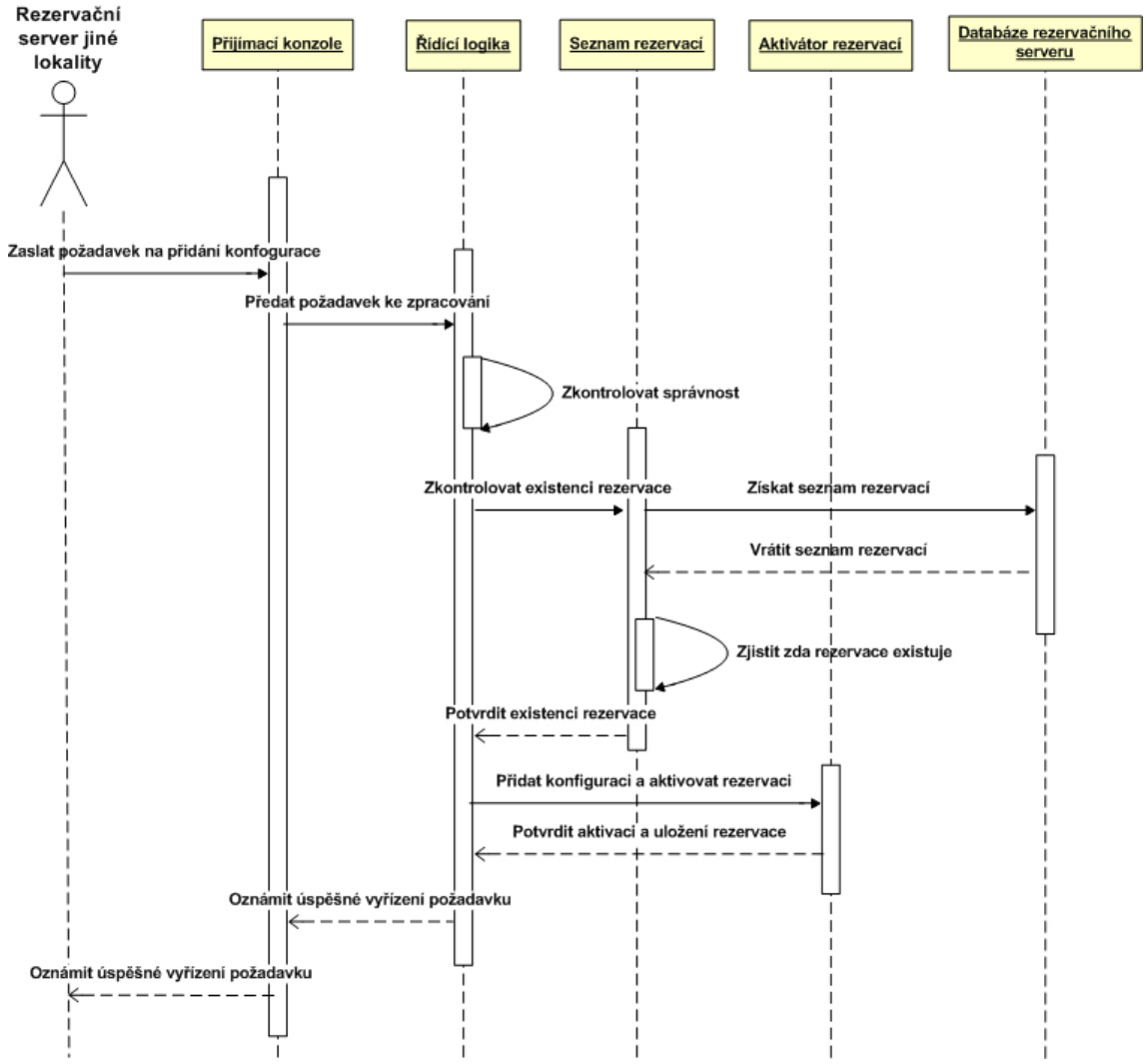
Řídící aplikace přidá konfiguraci k rezervaci. Konfiguraci potřebujeme, abychom věděli, jak budou zařízení společně spojena. Také nám slouží jako impuls k naplánování aktivačních skriptů. Grafické znázornění základního toku lze vidět na obrázku 7.

Primární aktéři

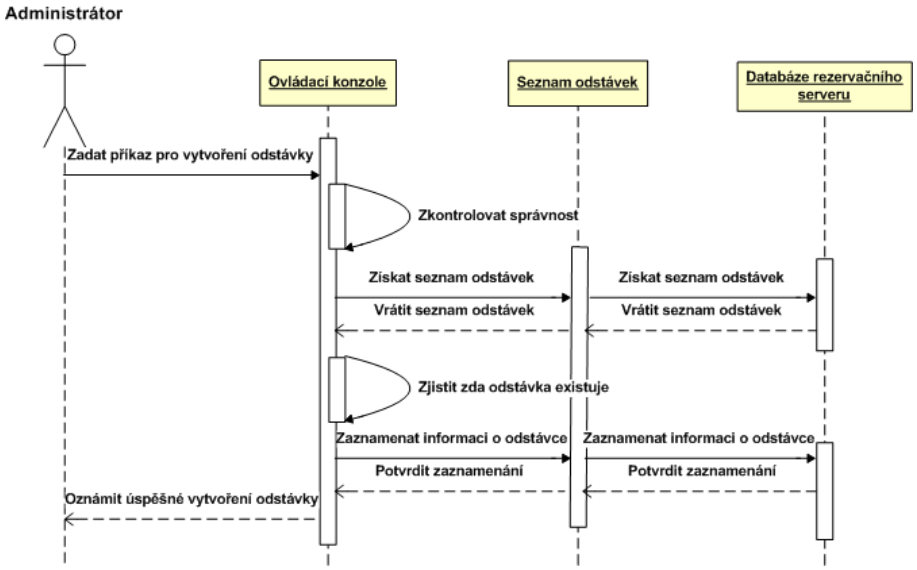
- Řídící aplikace

Základní tok událostí

1. Řídící aplikace zažádá o přidání konfigurace k rezervaci.
2. Rezervační server přijme požadavek pomocí definovaného Komunikačního protokolu a vytvoří nový požadavek.
3. Rezervační server zjistí, zdá má informaci o rezervaci, ke které je konfigurace určena.
4. Rezervační server přidá k rezervaci její konfiguraci a naplánuje aktivační skripty.
5. Rezervační server vrátí odpověď o úspěšném přidání konfigurace k rezervaci pomocí definovaného Komunikačního protokolu Řídící aplikaci.



Obrázek 7: Sekvenční diagram základního toku Přidání konfigurace k rezervaci



Obrázek 8: Sekvenční diagram základního toku Definování odstávky zařízení

4.7 Případ užití č.6 - Definování odstávky zařízení

Popis případu užití

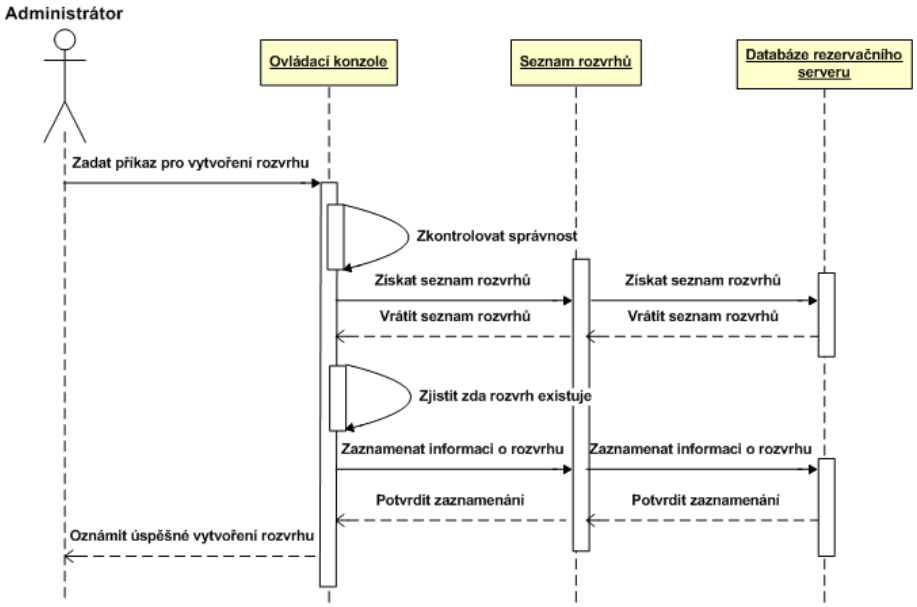
Administrátor definuje odstávku určitých anebo všech zařízení na určité časové období. Odstávky potřebujeme, pokud nechceme v učitém časovém období nabízet určitá zařízení. Grafické znázornění základního toku lze vidět na obrázku 8.

Primární aktéři

- Administrátor

Základní tok událostí

1. Administrátor zadá na ovládací konzoli příkaz pro tvorbu odstávky.
2. Rezervační server přijme příkaz a zkontroluje, zda je správně zapsaný.
3. Rezervační server zjistí, zda již neexistuje odstávka se stejným názvem.
4. Rezervační server si poznamená informaci o odstávce.
5. Rezervační server informuje administrátora o úspěšném vytvoření odstávky.



Obrázek 9: Sekvenční diagram základního toku Definování rozvrhu

4.8 Případ užití č.7 - Definování rozvrhu

Popis případu užití

Administrátor definuje časový rozvrh pro určitá nebo pro všechna zařízení na určité časové období. Časový rozvrh potřebujeme, pokud chceme efektivně určit hodiny, ve kterých budeme nabízet zařízení k rezervaci a naopak hodiny, ve kterých můžeme provádět údržbu a zařízení nepůjčovat. Grafické znázornění základního toku lze vidět na obrázku 9.

Primární aktéři

- Administrátor

Základní tok událostí

1. Administrátor zadá na ovládací konzoli příkaz pro tvorbu časového rozvrhu.
2. Rezervační server přijme příkaz a zkontroluje, zda je správně zapsaný.
3. Rezervační server zjistí, zda již neexistuje časový rozvrh se stejným názvem.
4. Rezervační server si poznamená informaci o časovém rozvrhu.
5. Rezervační server informuje administrátora o úspěšném vytvoření rozvrhu.

5 Rezervační server - Analýza a návrh

5.1 Architektura Rezervačního serveru

Architekura Rezervačního serveru se skládá ze tří komponent, viz. obrázek 10. Každá ze tří komponent slouží k jinému účelu a dohromady zajišťují kompletní funkčnost Rezervačního serveru. Toto tří-komponentní rozvržení se plně ztotožňuje s návrhovým vzorem MVC, který zajišťuje důsledné oddělení uživatelského rozhraní od aplikační logiky a od práce s daty. K této tří-komponentí arhitektuře dále přináleží dva balíčky, které obsahují třídy a pomocné funkce, které využívají všechny vrstvy architektury.

Komponenta číslo 1

Tato komponenta je jediná, se kterou uživatelé přímo komunikují. Stará se tedy primárně o zpracování a validaci požadavků na Razervační server. Tato data pak formátuje do instancí tříd, které reprezentují jednotlivé uživatelské požadavky a posílá je další komponentě ke zpracování. Další důležitou funkcí je přijímání výsledků o zpracování požadavků z další komponenty, zpracování těchto výsledků a prezentaci těchto výsledků uživatelům. Tato komponenta je reprezentována dvěma subsystémy. A to subystémem **Console_subsystem** a subsystémem **Accept_subsystem**.

Komponenta číslo 2

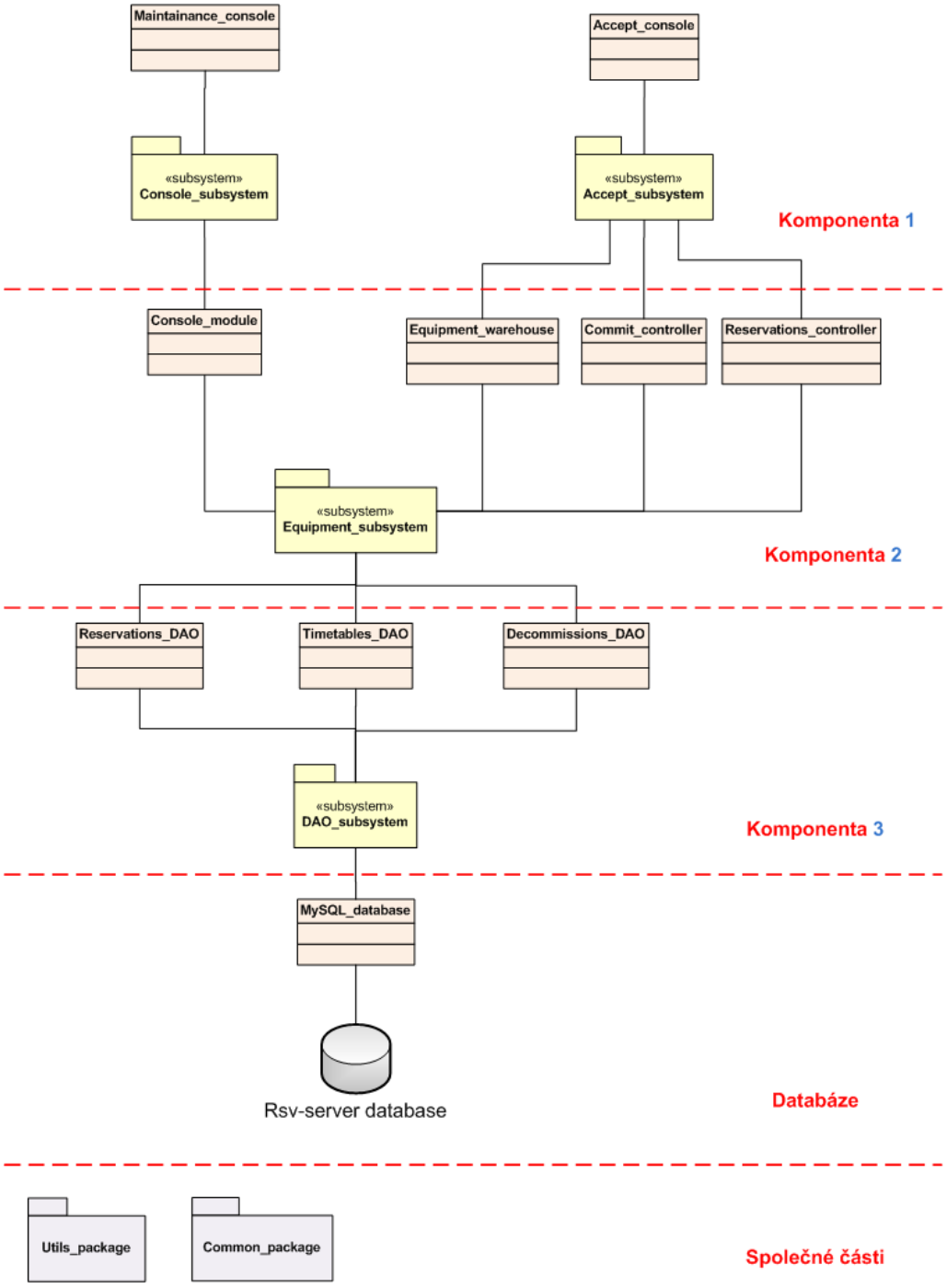
Tato komponenta se stará o zpracování požadavků, které přicházejí zaobalené v instancích tříd. Stará se taktéž o korektní zpracování paralelních požadavků, předání požadavků instancím tříd, kterým jsou určeny a v neposlední řadě taktéž o vzájemnou výměnu dat mezi komponentami. Taktéž odesílá zpracované požadavky zpět k prezentaci uživateli. Tato komponenta je reprezentována subsystémem **Equipment_subsystem**.

Komponenta číslo 3

Tato komponenta se stará o zpracování všech databázových požadavků, které přicházejí z jiných komponent. Ke zpracování těchto požadavků používá tzv. DAO objetky, kde každý z těchto objektů reprezentuje jednu z tabulek v databázi. Jedná se o jednu z metod ORM. Ke každému záznamu z tabulky tedy interně vytvoří instanci třídy, která ji reprezentuje a poskytne ji dalším komponentám k využívání. Veškeré změny, které jsou v této instanci třídy provedeny, pak zpětně promítá s použitím DAO objektů zpět do databáze. Tato komponenta je reprezentována subsystémem **DAO_subsystem**.

Společné části

K využití předchozích tří komponent se v Rezervačním serveru nacházají dva balíčky, které obsahují množiny tříd, které využívají všechny předchozí komponenty a taktéž sadu utilit pro zpracování dat. Tato komponenta je reprentována balíčky **Utils_package** a **Common_package**.



Obrázek 10: Celkový pohled na architekturu Rezervačního serveru

5.2 Subsystém Accept_subsystem

Tento subsystém, viz. obrázek 11, se stara přijímání požadavků na Rezervační server, např. požadavek na seznam zařízení v určitém časovém období, vytvoření rezervace a pod. Komunikuje s ním Řídící aplikace a Jiné rezervační servery. Přijaté požadavky jsou objektově zaobaleny a odeslány další komponentě pro zpracování. Po zpracování požadavku je odeslána opodvěď.

Třída: Accept_console

Tato třída je jednou ze tříd, skrze kterou lze komunikovat s Rezervačním serverem. Jejím hlavním úkolem je přijímat klienty skrze síťová spojení a vytvářet samostatná vlákna pro každého z klientů. Druhým a neméně důležitým úkolem je předat takto vytvořené uživatelské spojení instanci třídy Komunikačního protokolu **Communication_protocol**, která přijme veškerá data požadavku skrze definovaný Komunikační protokol. Z dat je pak vytvořeno objektové zaobalení požadavku, které je předáno instanci třídy **Request_splitter** ke zpracování.

Třída: Communication_protocol

Jedná se o abstraktní třídu reprezentující Komunikační protokol využívaný Rezervačním serverem. Dědičnost je využita z důvodů možné změny Komunikačního protokolu a je díky ní možné Komunikační protokol jednoduše vyměnit.

Třída: Communication_protocol_V1

Jedná se o třídu reprezentující konkrétně využívaný Komunikační protokol. Důležitou funkcí této třídy je parsování Komunikačního protokolu, validace dat a v neposlední řadě taktéž formátování odpovědi po zpracování požadavku. Výstupem z této třídy je objektové zaobalení požadavku, které reprezentuje jedna z instancí tříd typu **Request**.

Třída: Request_splitter

Úkolem této třídy je přijímat požadavky na Rezervační server a rozhodovat o způsobu jejich zpracování. Třída zná všechny lokality Virtuální laboratoře reprezentovány instancemi třídy **Locality**. Třída zná také místní lokalitu reprezentovanou instancí třídy **Local_locality** a využívají ji pro zpracování nedistribuovaných požadavků. Cizí lokality reprezentované instancemi třídy **Foreign_locality** a místní lokalita jsou pak společně využívány pro zpracování distribuovaných požadavků. Tento model zpracování dat je využitím návrhového vzoru *Observer*[3] neboli česky *Pozorovatel*. Třída využívá instanci třídy **Reservations_activator** ke zpracování požadavků na uložení konfigurace k rezervaci a následné aktivaci rezervace. K aktivaci rezervace je nutno znát jestli daná rezervace skutečně existuje. K tomuto využívá třída instanci třídy **Reservations_controller**.

[3]Detaily k veškerým použitým návrhovým vzorům lze nalézt v [8]

Třída: Reservations_activator

Jedná se o abstraktní třídu reprezentující momentálně využívanou metodu pro aktivování a deaktivování rezervace. Skrze tuto třídu lze přidat k rezervaci konfiguraci a následně rezervaci zaktivovat. Taktéž lze skrze tuto třídu aktivaci zrušit a zajistit spuštění deaktivační procedury.

Třída: Reservations_activator_V1

Jedná se o třídu reprezentující konkrétně využívanou metodu pro aktivaci rezervace. Třída aktivuje rezervace pomocí dvou speciálně vytvořených skriptů *activate.sh* a *deactivate.sh*, které reprezentují aktivační a deaktivačni proceduru. Spuštění skriptů se děje pomocí programu AT. Důležitou funkcí je taktéž přidávání konfigurace k rezervaci, které se děje pomocí uložení konfigurace do daného souboru, který je poté zpracováván aktivačním a deaktivačním skriptem.

Třída: Locality

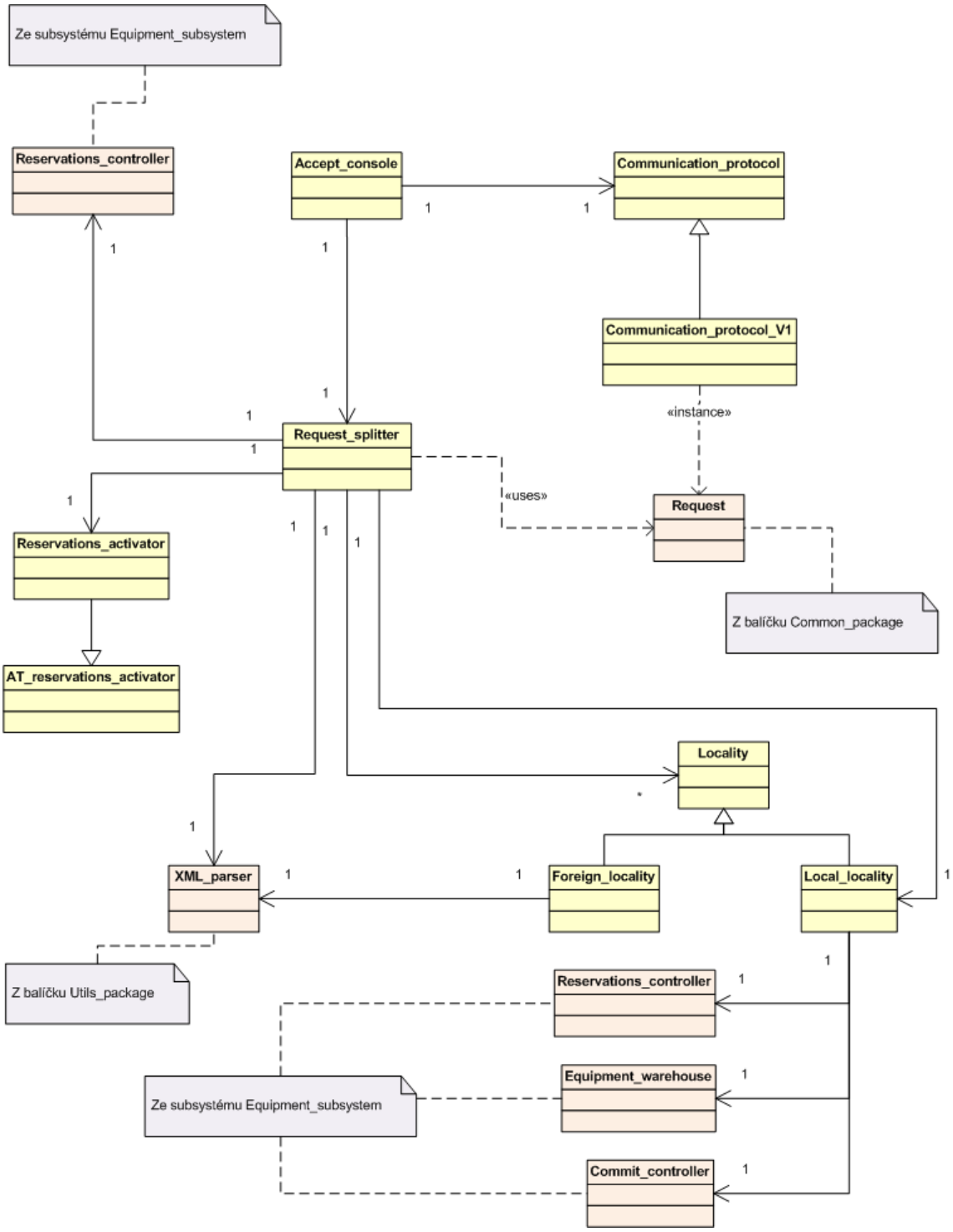
Jedná se o abstraktní třídu reprezentující lokalitu Virtuální laboratoře. Třída obsahuje metody pro zpracování veškerých požadavků, které může rezervační server zpracovávat. Každá z těchto metod vrací taktéž zprávu a úspěchu nebo neúspěchu provedeného požadavku.

Třída: Foreign_locality

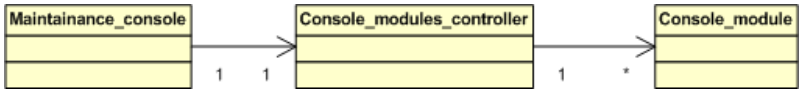
Jedná se o třídu reperezentující cizí lokalitu Virtuální laboratoře. Dědí ze třídy **Locality** a implementuje veškeré zpracování požadavků, které jsou určené pro cizí lokalitu. K práci s XML využívá instanci třídy **XML_parser**. Pokud se není možné připojit k cizí lokalitě, pak každý z požadavků končí chybou.

Třída: Local_locality

Tato třída reprezentuje místní lokalitu Virtuální laboratoře. Dědí ze třídy **Locality** a implementuje veškeré zpracování požadavků, které jsou určené pro místní lokalitu. Rozhoduje se, jak zpracovat každý z požadavků, které přicházejí na Rezervační server. Ke zpracování požadavků na získání nebo rezervaci zařízení využívá instanci třídy **Equipment_warehouse**, ke zpracování požadavků na potvrzení rezervace využívá instanci třídy **Commit_controller** a v poslední řadě k mazání rezervací využívá instanci třídy **Reservations_controller**.



Obrázek 11: Architektura subsystému Accept_subsystem



Obrázek 12: Architektura subsystému Console_subsystem

5.3 Subsystém Console_subsystem

Tento subsystém, viz. obrázek 12, se stará o zpracování požadavků z administrátorské konzole. S tímto subsystémem komunikuje výhradně administrátor a upravuje pomocí něho chování Rezervačního serveru.

Třída: Maintainance_console

Skrze tuto třídu jsou přijímána administrátorská spojení. Třída pro každé administrátorské spojení vytváří nové samostatné vlákno. Po přijmutí nového uživatelského spojení je administrátorovi zobrazena konzole se seznamem příkazů, který vzniká na základě komunikace s instancí třídy **Console_modules_controller**. Po přijmutí příkazu z konzole je tento příkaz zaslán instanci třídy **Console_modules_controller** pro další zpracování. Po zpracování požadavku je odpověď zaslána zpět administrátorovi.

Třída: Console_modules_controller

Tato třída slouží ke zpracování požadavků přicházejících z instance třídy **Maintainance_console**. Jedná se o textové příkazy, které jsou zpracovány a následně jsou předány každému z modulů, které jsou reprezentovány instancí třídy **Console_module**. Pokud modul dokáže příkaz zpracovat, pak oznámí, že dokáže daný příkaz zpracovat. Tento modul je implementací návrhového vzoru *Chain-of-responsibility*, česky *Řetěz zodpovědností*. Kvůli možnosti mít připojeno více administrátorů, tato třída taktéž řeší serializaci příkazů.

Třída: Console_module

Tato abstraktní třída reprezentuje modul administrátorské konzole. Obsahuje metodu pro zpracování příkazů a zjištění informací o dostupných příkazech. Třída, která dědí z této třídy, může být později využita jako jeden z modulů, administrátorské konzole.

5.4 Subsystém Equipment_subsystem

Tento subsystém, viz. obrázek 13, se stará o zpracování požadavků na Rezervační server. Subsystém obsahuje instance tříd, které jsou schopny zpracovat každý z příchozích požadavků. Subsystém se také stará o korektní provádění distribuovaných požadavků a řeší tudíž veškeré paralelní operace.

Třída: Equipment_warehouse

Tato třída se stará o správu zařízení, které může Rezervační server poskytovat. Obsahuje seznam všech zařízení, a to jak v podobě XML reprezentace, tak v objektové podobě, kterou reprezentují instance třídy **Device**. Třída využívá také instance třídy **Equipment_restriction**, které slouží k omezení množiny dostupných zařízení. Tato architektura je využitím návrhového vzoru *Observer*, neboli česky *Pozorovatel*. V případě požadavku na seznam nebo rezervaci zařízení, kontaktuje třída každou z instancí třídy **Equipment_restriction** a zjišťuje, které ze zařízení *nemůže poskytnout* v daném časovém období. Odpovědi ze všech tříd jsou pak spojeny a výsledná množina zařízení neobsahuje zařízení, které nelze poskytnout. Po úspěšném vytvoření rezervace kontaktuje tato třída instanci třídy **Commit_controller** a vytváří pomocí ní rezervaci, která čeká na potvrzení.

Třída: Device

Tato třída je objektovou reprezentací zařízení, které může rezervační server poskytovat. Vzniká na základě zpracování XML popisu zařízení. V současné podobě stačí Rezervačnímu serveru znát pouze XML popis zařízení. Třída proto není v současné době využívána a je připravena pro možné pozdější využití.

Třída: Equipment_restriction

Tato abstraktní třída reprezentuje třídu, která omezuje množinu zařízení, kterou může Rezervační server v daném časovém období poskytnout. Kvůli zpracování paralelních požadavků řeší třída taktéž zamykání a serializaci.

Třída: Commit_controller

Tato třída se stará o správu rezervací, které doposud nebyly potvrzeny. Tato třída dostává informace od instance třídy **Equipment_warehouse** a vytváří rezervaci, která čeká na potvrzení. Třída obsahuje samostatné vlákno, které každou vteřinu kontroluje všechny rezervace, na které nepřišlo potvrzení a pokud na rezervaci nepřišlo potvrzení do dvou minut, pak je tato rezervace bez potvrzení smazána. Třída interně vytváří instance třídy **Commit**, které reprezentují rezervaci, na kterou zatím nepřišlo potvrzení. Po příchodu potvrzení předá třída informace instanci třídy **Reservations_controller**, která vytvoří novou rezervaci. Třída dědí ze třídy **Equipment_restriction** s slouží tedy jako omezení dostupných zařízení.

Třída: Reservations_controller

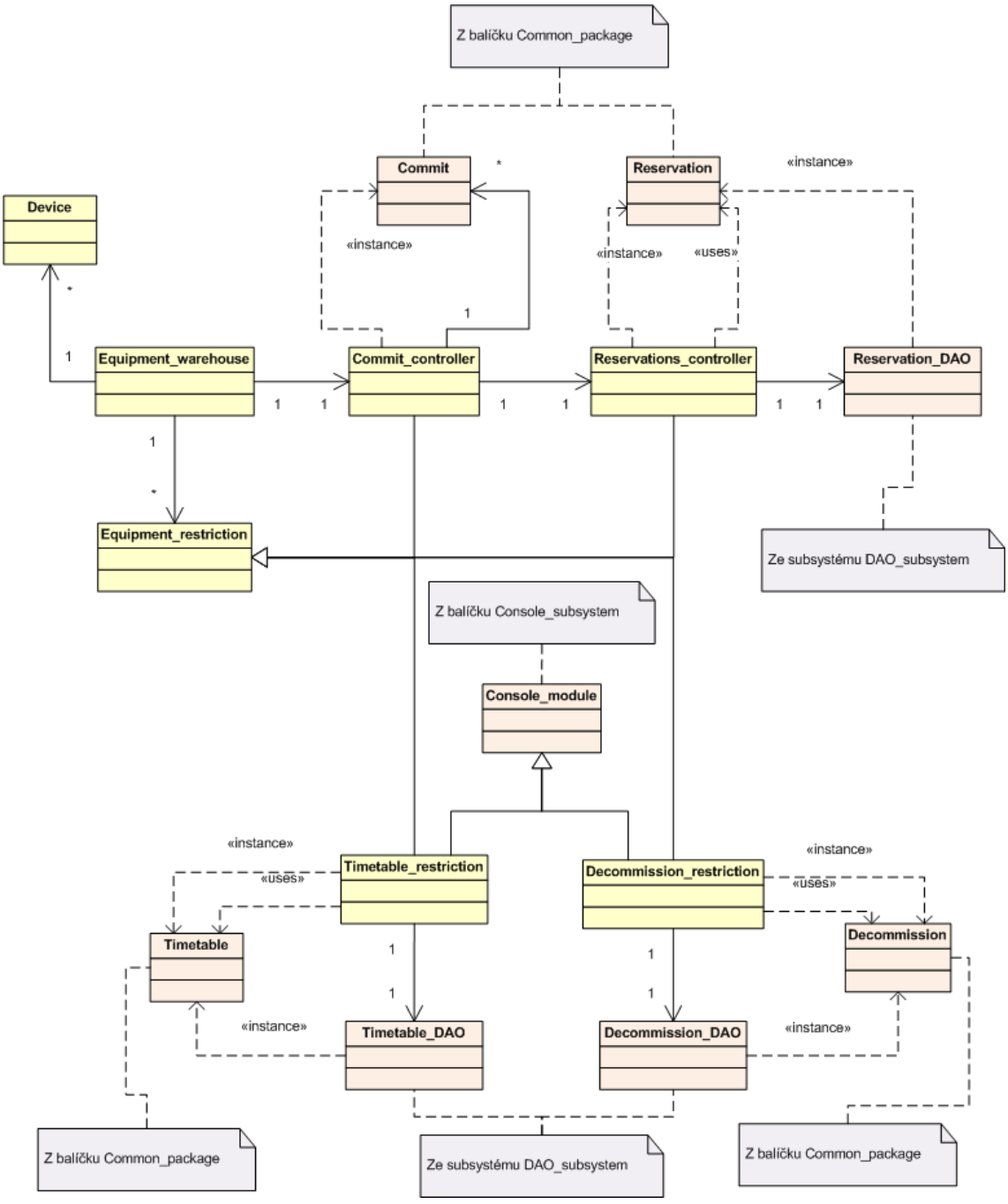
Tato třída se stará o správu rezervací, na které již přišlo potvrzení a byly vytvořeny. Třída využívá instanci třídy **Reservation_DAO** pro všechny databázové operace, které se týkají rezervací. Ke své práci využívá instanci třídy **Reservation**, která reprezentuje vytvořenou rezervaci. Tuto třídu vytváří sama třída anebo instance třídy **Reservation_DAO**. Třída dědí ze třídy **Equipment_restriction** s slouží tedy jako omezení dostupných zařízení.

Třída: Decommission_restriction

Tato třída slouží ke správě odstávek zařízení. Odstávka zařízení je vytvářena administrátorem, pokud je potřeba odstavit na určité časové období některá zařízení a neposkytovat je pro rezervace. Třída využívá instanci třídy **Decommission_DAO** ke všem databázovým operacím tykajících se odstávek. Jako třída dědící z třídy **Console_Module** může tato třída představovat jeden z modulů ovládací konzole a získávat tak informace z ovládací konzole. Ke své práci využívá instanci třídy **Decommission**, která reprezentuje odstávku. Tuto třídu vytváří sama třída anebo instance třídy **Decommission_DAO**. Třída taktéž dědí ze třídy **Equipment_restriction** a slouží tedy jako omezení dostupných zařízení.

Třída: Timetable_restriction

Tato třída slouží ke správě rozvrhů pro zařízení. Rozvrh pro zařízení je vytvářen administrátorem, pokud je potřeba vytvořit rozvrh pro zařízení na určité časové období a poskytovat zařízení jen v určitých časových oknech, specifikovaných v rozvrhu. Třída využívá instanci třídy **Timetable_DAO** ke všem databázovým operacím tykajících se rozvrhů pro zařízení. Jako třída dědící z třídy **Console_Module** může tato třída představovat jeden z modulů ovládací konzole a získávat tak informace z ovládací konzole. Ke své práci využívá instanci třídy **Timetable**, která reprezentuje rozvrh pro zařízení. Tuto třídu vytváří sama třída anebo instance třídy **Timetable_DAO**. Třída taktéž dědí ze třídy **Equipment_restriction** a slouží tedy jako omezení dostupných zařízení.



Obrázek 13: Architektura subsystému Equipment_subsystem

5.5 Subsystém DAO_subsystem

Tento subsystém, viz. obrázek 14, se stará o veškeré databázové požadavky a jako jediný ze všech subsystémů komunikuje s databází. Ke zpracování databázovýxch požadavků používá tzv. DAO objekty. Tj. objekty, které zastupují jednotlivé tabulky v databázi. Jedná se o využití návrhového vzoru *DAO - Database Acces Object* česky *Objekt pro přístup do databáze*. DAO je jeden ze způsobů ORM - Object relations mapping.

Třída: Reservation_DAO

Tato abstraktní třída reprezentuje objekt, pomocí kterého lze provádět operace nad rezervacemi. Tj. lze pomocí něj uložit rezervaci do databáze, načíst ji z databáze a pod. Ke své práci využívá třída instanci třídy **Reservation**, která představuje objektové zapouzdření rezervace.

Třída: Decommission_DAO

Tato abstraktní třída reprezentuje objekt, pomocí kterého lze provádět operace nad odstávkami. Tj. lze pomocí něj uložit odstávku do databáze, načíst ji z databáze a pod. Ke své práci využívá třída instanci třídy **Decommission**, která představuje objektové zapouzdření odstávky.

Třída: Timetable_DAO

Tato abstraktní třída reprezentuje objekt, pomocí kterého lze provádět operace nad rozvrhy pro zařízení. Tj. lze pomocí něj uložit rozvrh pro zařízení do databáze, načíst jej z databáze a pod. Ke své práci využívá třída instanci třídy **Timetable**, která představuje objektové zapouzdření rozvrhu pro zařízení.

Třída: MySQL_database

Tato třída reprezentuje spojení na MySQL databázi, která je v současné verzi použita jako hlavní databáze. Pomocí této třídy mohou ostatní DAO objekty komunikovat s databází, zjistit, zda je spojení funkční a pod.

Třída: MySQL_reservation_DAO

Tato třída dědí ze třídy **Reservation_DAO** a implementuje její veškerou potřebnou funkčnost. Jelikož je objekt rezervace tvořen jak samotnou rezervací tak seznamem zařízení, která jsou zarezervována využívá třída instanci třídy **MySQL_reserved_devices_DAO**, pomocí které pracuje se seznamem zařízení v dané rezervaci. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída instanci třídy **MySQL_database**.

Třída: MySQL_reserved_devices_DAO

Tato třída zajišťuje veškerou potřebnou funkčnost, která je potřeba pro práci se seznamem zarezervovaných zařízení. Tato třída je využívána instancí třídy **MySQL_reservation_DAO**, od které dostává požadavky pro práci se seznamem zarezervovaných zařízení. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída instanci třídy **MySQL_database**.

Třída: MySQL_decommission_DAO

Tato třída dědí ze třídy **Decommission_DAO** a implementuje její veškerou potřebnou funkčnost.Jelikož je objekt odstávky tvořen jak samotnou odstávkou tak seznamem zařízení, která jsou v odstávce, využívá třída instanci třídy **MySQL_decommissioned_devices_DAO**, pomocí které pracuje se seznamem zařízení v dané odstávce. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída třídu **MySQL_database**.

Třída: MySQL_decommissioned_devices_DAO

Tato třída zajišťuje veškerou potřebnou funkčnost, která je potřeba pro práci se seznamem zařízení, která jsou v odstávce. Tato třída je využívána instancí třídy **MySQL_decommission_DAO**, od které dostává požadavky pro práci se seznamem zařízení, která jsou v odstávce. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída třídu **MySQL_database**.

Třída: MySQL_timetable_DAO

Tato třída dědí ze třídy **Timetable_DAO** a implementuje její veškerou potřebnou funkčnost. Jelikož je objekt rozvrhu pro zařízení tvořen jak samotným rozvrhem tak seznamem zařízení, na která se rozvrh vztahuje a taktéž záznamy v rozvrhu, využívá třída instanci třídy **MySQL_devices_in_timetable_DAO**, pomocí které pracuje se seznamem zařízení, které přináleží danému rozvrhu a instanci třídy **MySQL_entries_DAO**, pomocí které pracuje se záznamy v rozvrhu. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída třídu **MySQL_database**.

Třída: MySQL_devices_in_timetable_DAO

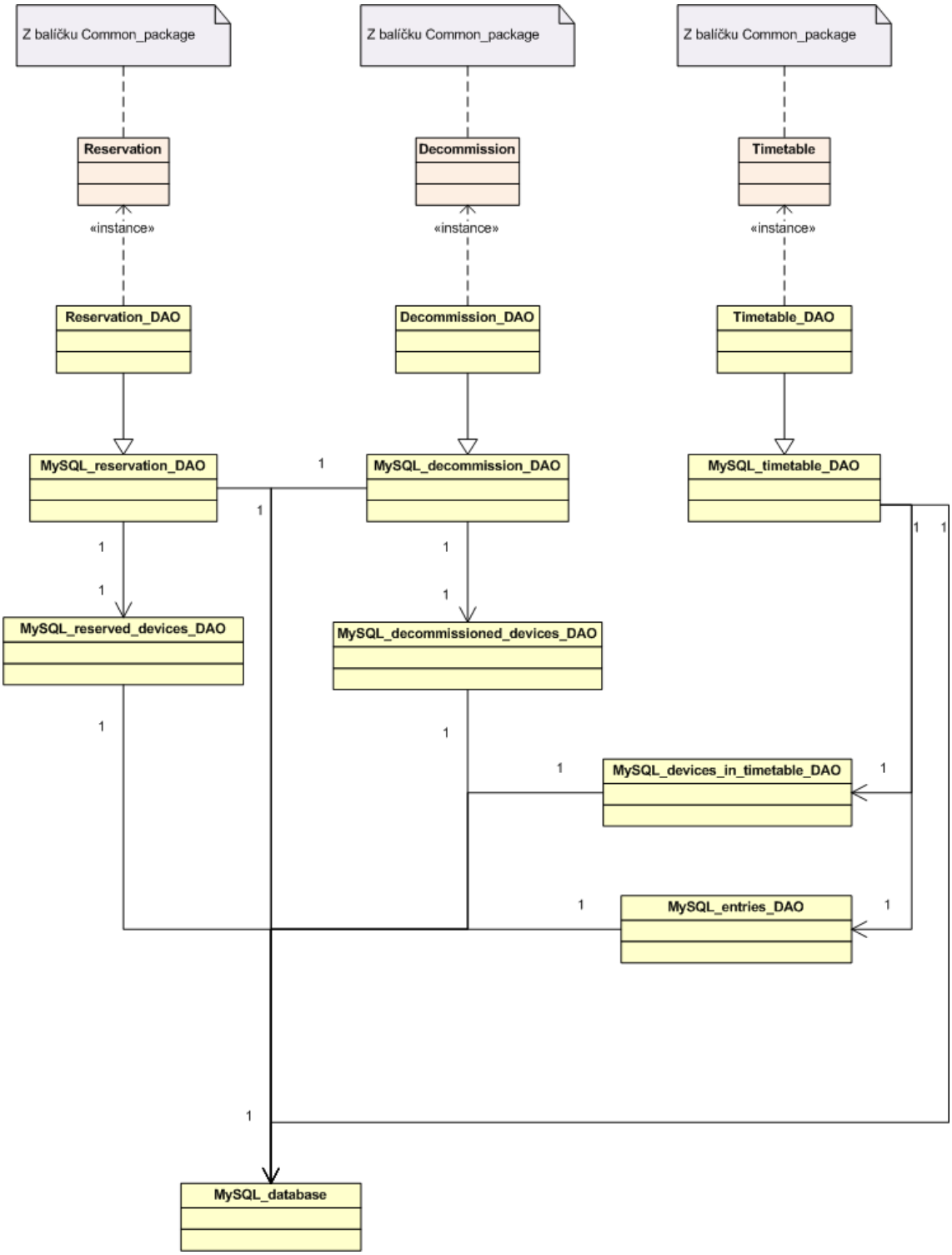
Tato třída zajišťuje veškerou potřebnou funkčnost, která je potřeba pro práci se seznamem zařízení, která patří k rozvrhu zařízení. Tato třída je využívána instancí třídy **MySQL_timetable_DAO**, od které dostává požadavky pro práci se seznamem zařízení, která jsou v rozvrhu pro zařízení. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída třídu **MySQL_database**.

Třída: MySQL_entries_DAO

Tato třída zajišťuje veškerou potřebnou funkčnost, která je potřeba pro práci se záznamy v rozvrhu pro zařízení. Tato třída je využívána instancí třídy **MySQL_timetable_DAO**, od které dostává požadavky pro práci se seznamem záznamů, které jsou v rozvrhu pro zařízení. Interně jsou všechny databázové operace realizovány formou *předpřipravených dotazů*, které výrazně urychlují provedení databázových operací. Pro práci s databází využívá třída třídu **MySQL_database**.

Sdílení spojení

Instance tříd **MySQL_devices_in_timetable_DAO**,**MySQL_entries_DAO** a **MySQL_timetable_DAO** sdílí stejné spojení na databázi stejně jako instance třídy **MySQL_reservation_DAO**,**MySQL_reserved_devices_DAO** a instance tříd **MySQL_decommission_DAO**, **MySQL_decommissioned_devices_DAO**. Instance tříd sdílejí společné připojení na databázi, aby bylo možné uchovat ID transakce mezi jednotlivými databázovymi operacemi a korektně na ně reagovat.



Obrázek 14: Architektura subsystému DAO_subsystem

5.6 Balíček Common_package

V tomto balíčku, viz. obrázek 15, se nacházejí třídy, které jsou globálně využívány více komponentami. Najdeme zde třídy, které zaobalují požadavky na Rezervační server nebo třídy reprezentující jednotlivé objekty, se kterými v Rezervačním serveru zacházíme. Například třídu reprezentující rezervaci, rozvrh pro zařízení a pod.

Třída: Request

Tato abstraktní třída je objektovým zaobalením požadavku na Rezervační server. Sama o sobě nenabízí žádnou fukčnost, jelikož každý z požadavků je v něčem odlišný a není možné nalézt žádné společné rysy. Třída proto slouží hlavně k logickému ujasnění, že zde existují požadavky, a že každý požadavek na Rezervační server je speciálním případem požadavku.

Třída: Get_offer_request

Tato třída představuje objektové zaobalení požadavku na seznam zařízení. Obsahuje tedy všechny potřebné informace, které potřebuje Rezervační server znát, aby mohl zobrazit seznam dostupných zařízení. Třída taktéž dědí ze třídy **Request** a je tedy požadavkem na Rezervační server.

Třída: Attach_request

Tato třída představuje objektové zaobalení požadavku na přidání konfigurace k rezervaci. Obsahuje tedy všechny potřebné informace, které potřebuje Rezervační server znát, aby mohl přidat konfiguraci k rezervaci. Třída taktéž dědí ze třídy **Request** a je tedy požadavkem na Rezervační server.

Třída: Cancel_request

Tato třída představuje objektové zaobalení požadavku na smazání existující rezervace nebo rezervace čekající na potvrzení. Obsahuje tedy všechny potřebné informace, které potřebuje Rezervační server znát, aby mohl zrušit rezervaci. Třída taktéž dědí ze třídy **Request** a je tedy požadavkem na Rezervační server.

Třída: Commit_request

Tato třída představuje objektové zaobalení požadavku na potvrzení rezervace. Obsahuje tedy všechny potřebné informace, které potřebuje Rezervační server znát, aby mohl potvrdit rezervaci. Třída taktéž dědí ze třídy **Request** a je tedy požadavkem na Rezervační server.

Třída: Reserve_request

Tato třída představuje objektové zaobalení požadavku na vytvoření rezervace. Obsahuje tedy všechny potřebné informace, které potřebuje Rezervační server znát, aby mohl vytvořit rezervaci. Třída taktéž dědí ze třídy **Request** a je tedy požadavkem na Rezervační server.

Třída: Initial_data

Tato třída obsahuje informace o uživatelském spojení. Tj. z jaké IP adresy a TCP portu požadavek přišel, ID vlákna a pod. Tyto informace nejsou v této verzi zatím využívány. Což se však může v příštích verzích změnit.

Třída: Reservation

Tato třída je objektovým zaobalením rezervace a obsahuje všechny potřebné informace o rezervaci. Tj. obsahuje její ID, čas od - do, seznam zařízení a pod. Tato třída je využívána v komponentách Rezervačního serveru při práci s rezervacemi. Třídy **Reservation** a **Commit** jsou stejné, a tudíž žádná z nich není specializací té druhé. Různé názvy slouží hlavně k logickému oddělení, v jakém stavu se zřovna rezervace nachází.

Třída: Commit

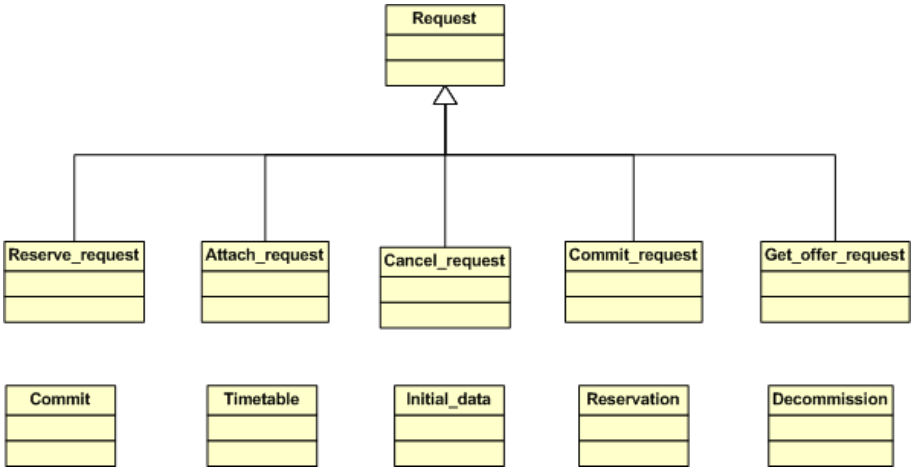
Tato třída je objektovým zaobalením rezervace, která čeká na potvrzení a obsahuje všechny potřebné informace o rezervaci. Tj. obsahuje její ID, čas od - do, seznam zařízení a pod. Tato třída je využívána v komponentách Rezervačního serveru při práci s nepotvrzenými rezervacemi. Třídy **Reservation** a **Commit** jsou stejné, a tudíž žádná z nich není specializací té druhé. Různé názvy slouží hlavně k logickému oddělení, v jakém stavu se zřovna rezervace nachází.

Třída: Timetable

Tato třída je objektovým zaobalením rozvrhu pro zařízení a obsahuje všechny potřebné informace o rozvrhu pro zařízení. Tj. obsahuje jeho ID, čas od - do, seznam zařízení, seznam záznamů v rozvrhu, uživatelskou skupinu a pod. Tato třída je využívána komponentami Rezervačního serveru při práci s rozvrhy pro zařízení.

Třída: Decommission

Tato třída je objektovým zaobalením odstávky na zařízení a obsahuje všechny potřebné informace o odstávce na zařízení. Tj. obsahuje její ID, čas od - do, seznam zařízení, uživatelskou skupinu a pod. Tato třída je využívána komponentami Rezervačního serveru při práci s odstávkami na zařízení.



Obrázek 15: Architektura balíčku Common_package

5.7 Balíček Utils_package

V tomto balíčku, viz. obrázek 16, se nacházejí třídy pro zajištění doplňkové funkčnosti. Najdeme zde například parser XML souborů, třídu pro práci s prostředky, třídu pro logování a pod.

Třída: Messages

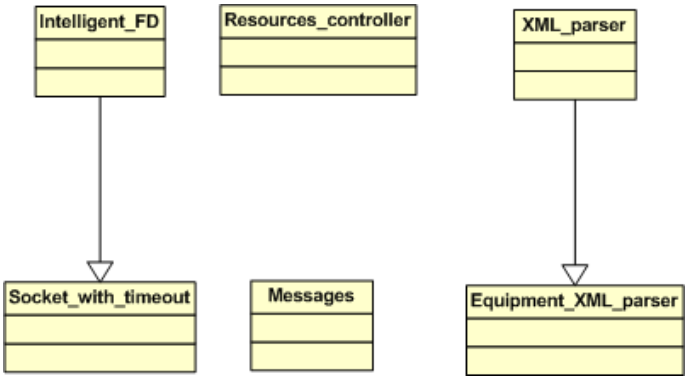
Jedná se o třídu sloužící k logování událostí do systému Syslog. Třída dodržuje logovací konvenci systému Virtlab, je globálně přístupná a může jí využít kterýkoliv modul, který má potřebu zalogovat některou z významných událostí.

Třída: Resources_controller

Tato třída se stará o správu dostupných zdrojů, které jsou přiděleny Rezervačnímu serveru. V současné době slouží k uchovávání počtu vláken. Jelikož jsou vlákna tvořena na více místech, je potřeba zaznamenat každé vytvořené nebo smazané vlákno a v případě dosažení maximálního počtu vláken, další vlákna nevytvářet. Hodnota by se dala přečíst i z OS, ale nenabízela by ochranu proti souběžným přístupům jako tato třída.

Třída: Intelligent_FD

Tato abstraktní třída reprezentuje množinu inteligentních FD. Obsahuje návratové kódy a předpisy funkcí, které mají inteligentní FD používat. V Rezervačním serveru je jediný inteligentní FD socket s možností nastavení vypršení časového limitu pro příjem dat.



Obrázek 16: Architektura balíčku Utils_package

Třída: Socket_with_timeout

Tato třída reprezentuje socket s možností nastavení vypršení časového limitu pro příjem dat. V Rezervačním serveru je toto jediná třída, která dědí ze třídy **Socket_with_timeout**. Tato třída je využívána každým z modulů, který potřebuje komunikovat přes síť a zároveň mít možnost specifikovat, jak dlouho čekat na příjem dat.

Třída: XML_parser

Tato abstraktní třída reprezentuje parser XML dat specifického formátu, který je využit v Rezervačním serveru. Tato třída je pouze obecným předpisem, jak by měl parser XML dat vypadat. Aktuální verzi XML parseru implementuje třída, která tuto třídu dědí.

Třída: Equipment_XML_parser

Tato třída dědí ze třídy **XML_parser** a figuruje jako parser XML dat využívaných v Rezervačním serveru. Implementuje současnou podobu XML dat a veškeré operace nad XML daty zachovávají tuto podobu. Tato třída je používána komponentami Rezervačního serveru všude, kde je potřeba pracovat s XML daty.

6 Rezervační server - Implementace

Rezervační server byl implementován s dodržením zásad Softwarového inženýrství. Blíže jsou zde popsány nejdůležitější problémy, které bylo nutné při implementaci vyřešit.

6.1 Zamykání omezujících podmínek

Jelikož náš server využívá parelelní a distribuované zpracování požadavků a také umožňuje aministrátorovi za běhu měnit chování Rezervačního serveru, je velice důležité zajistit korektní zamykání omezujících podmínek, jelikož se v nich pracuje s daty, která budou poskytnuta uživateli. Pokud bychom korektně neuzamykali a neodemykali omezující podmínky, pak by docházelo k nekonzistencím v množině zařízení, která nelze poskytnout. Mohlo by se například stát, že by administrátor definoval odstávku v době, kdy se vytváří rezervace a zařízení, která by měla být v odstávce, by se nakonec dostala i do nově vznikající rezervace.

Zamykací protokol lze popsat následovně:

- **Pokus se zamknout všechny omezující podmínky.**
- **Pokud se nepodaří všechny omezující podmínky zamknout, pak zamknuté omezující podmínky odemkni, počkej náhodnou dobu a zkus to znovu.**

Je jasné, že nemůžeme používat zamykání, u kterého by se při neúspěšném pokusu vlákno zablokovalo a čekalo na uvolnění omezující podmínky. V takovém případě by nastalo uváznutí. Proto používáme méně striktní verzi zamykání, které se pouze pokusí danou omezující podmínku uzamknout a v případě neúspěchu vrátí chybovou hlášku. Tato méně striktní verze zamykání je relizována systémovou funkcí *pthread_mutex_trylock*[4].

6.2 Práce s vlákny

Rezervační server pracuje s vlákny velice jednouše. Každému příchozímu požadavku je vytvořeno nové vlákno, které se stará o zpracování požadavku. Po zpracování požadavku vlákno zaniká. Rezervační server tak může paralelně obsloužit více klientů.

Nové vlákno je taktéž vytvářeno pro připojení na administrátorskou konzoli. Po odhlášení z konzole vlákno zaniká.

[4] Více informací o synchronizaci procesů lze nalézt na [3], detaily k systémovým voláním lze nalézt v [6]

6.3 Přidání dalších příkazů/požadavků

Rezervační server obsahuje možnost rozšíření množiny příkazů/požadavků. Pro přidání nového příkazu/požadavku musí programátor provést následující posloupnost kroků:

1. Dědit ze třídy **Request**
2. Přidat do Komunikačního protokolu reprezentovaného třídou **Communication_protocol_V1**, popis, jak daný požadavek vypadá, jaké má parametry a jaká je na něj definovaná odpověď.
3. Přidat funkci pro zpracování požadavku do třídy **Request_splitter**, ve které se definuje pořadí kroků, které se musí provést v distribuované nebo nedistribuované verzi požadavku.
4. Pokud je požadavek distribuovaný, pak musí nový požadavek přidat do třídy **Foreign_locality**, jak má vypadat odeslání požadavku jinému Rezervačnímu serveru a taktéž, jak má probíhat zpracování odpovědi.
5. Přidat nový požadavek do třídy **Local_locality** způsob, jakým je požadavek Rezervačním serverem zpracován.

6.4 Implementace databázových operací

Rezervační server často využívá databázové operace a je tudíž nutné zajistit, aby tyto operace probíhaly co nejrychleji. Všechny databázové operace jsou tudíž implementovány jako *předpřipravené dotazy*, které umožňují rychlejší provádění dotazu. Předpřipravený dotaz je zpracován databázovým systémem a najde se nejrychlejší způsob, jak tento dotaz provést. Tj. stejná procedura jako u kteréhokoliv jiného databázového dotazu. Avšak takto připravený dotaz je poté uložen a od klienta již pouze přicházejí parametry, které jsou později do dotazu vloženy. Odpadá nám tedy nutnost při každém provedení dotazu tento dotaz kompilovat a testovat jeho validitu.

6.5 Dvoufázový potvrzovací protokol

Dvoufázový potvrzovací protokol implementujeme kvůli distribuované a paralelní architektuře Virtuální laboratoře a umožňuje nám bezpečné vytváření rezervací.

Rezervace je vytvořena následujícím způsobem:

1. Zjisti, ze kterých lokací pocházejí zařízení v dané rezervaci.
2. Kontaktuj každou z těchto lokalit požadavkem na vytvoření rezervace, který obsahuje jen zařízení z této lokality.
3. V každé z lokalit zkontroluj, zda jsou tato zařízení v dané době dostupná a pokud ano, tak na ně vytvoř rezervaci čekající na potvrzení. Pokud ne, pak ohlaš chybu.
4. Pokud proběhly všechny rezervace úspěšně, pak každé z lokalit pošli příkaz pro potvrzení rezervace. Pokud ne, pak pošli každé lokalitě příkaz pro zrušení rezervace.
5. V každé z lokalit potvrď rezervaci. Pokud se to nepovede, pak ohlaš chybu.
6. Pokud proběhla všechna potvrzení úspěšně, pak oznam úspěšné vytvoření rezervace. Pokud ne, pak pošli každé lokalitě příkaz pro zrušení rezervace.

Dvoufázový potvrzovací protokol přichází na řadu ve chvíli, kdy se čeká na potvrzení rezervace a zajistí, že pokud se neprovede rezervace ve všech lokalitách, ve kterých jsou zařízení dané rezervace, pak se daná rezervace neprovede vůbec. Pokud dojde kdykoliv při zpracování k výpadku komunikace, pak rezervaci nevytvoříme a ohlásíme chybu. V každé z lokalit se provádí stejná posloupnost kroků, jaká je popsaná v případech užití.

7 Konzolový server

Konzolový server se stará o zpřístupnění konzole zařízení uživateli. Uživatel komunikuje s Konzolový serverem pomocí Java appletu nebo pomocí standartního telnet klienta. Konzolový server pak komunikuje s konzolí zařízení skrze TCP spojení, sériovou linku nebo slouží pouze jako proxy mezi uživatelem a Konzolovým serverem cizí lokality. Taktéž se stará o autentizaci uživatele a kontrolu zakázaných příkazů. Pokud je uživatel tutor, pak je ostatním uživatelům na konzoli zařízení zaslána notifikace o připojení tutora.

7.1 Terminologie Konzolového serveru

U Konzolového serveru budeme používat následující terminologii:

- **Proxy režim** - Jedná se o režim Konzolového serveru, ve kterém Konzolový server slouží pouze jako proxy server a přeposílá uživatelská data Konzolovému serveru jiné lokality.
- **Autentizace uživatele** - Konzolový server autentizuje každého uživatele, který přistupuje na konzoli zařízení. Je to z důvodu, že přístup k zařízením smí mít pouze autorizované osoby.
- **Konzole zařízení** - Jedná se o konzoli, skrze kterou lze s daným zařízením komunikovat. Nejčastěji je tato konzole dostupná skrze sériový port nebo TCP spojení.
- **Filtry** - Provoz mezi uživatelem a zařízením je třeba filtrovat a to z důvodu, aby nemohli uživatelé vkládat na zařízení příkazy, které by mohli způsobit poškození zařízení.
- **Student** - Jedná se o uživatele Konzolového serveru, který reprezentuje studenta a nemá možnost ovlivňovat chování konzole zařízení pro ostatní uživatele.
- **Tutor** - Jedná se o uživatele Konzolového serveru, který reprezentuje tutora a má možnost ovlivňovat chování konzole zařízení pro ostatní uživatele.
- **Stav filtru** - Jedná se stav, ve kterém se nachází jeden z filtrů využívaný Konzolovým serverem. Například se může uložit právě napsaný řádek na konzoli zařízení, aby se vědělo, co uživatel stihl zapsat na konzoli předtím než se odpojil.

8 Konzolový server - Specifikace požadavků

8.1 Definice jednotlivých aktérů

S Konzolovým serverem pracují tři druhy aktérů. Každý z nich má určité očekávání a cíle, které potřebuje splnit, viz. obrázek 17. Aktéři jsou tedy následující:

Uživatel

Jako aktéra Uživatel bereme uživatele lokality Virtuální laboratoře, který se potřebuje připojit na konzoli daného zařízení. Uživatel má dvě možnosti, jak komunikovat s Konzolovým serverem. První a preferovaná z nich je komunikace skrze webový Java applet poskytovaný Virtuální laboratoří. Preferován je proto, že úvodní komunikace s Konzolovým server probíhá automaticky a Uživatel nemusí mít žádný pojem o Komunikačním protokolu Konzolového serveru. Druhou možnosí je využití standardního Telnet klienta. V tomto případě však již musí uživatel na začátku využít definovaný Komunikační protokol.

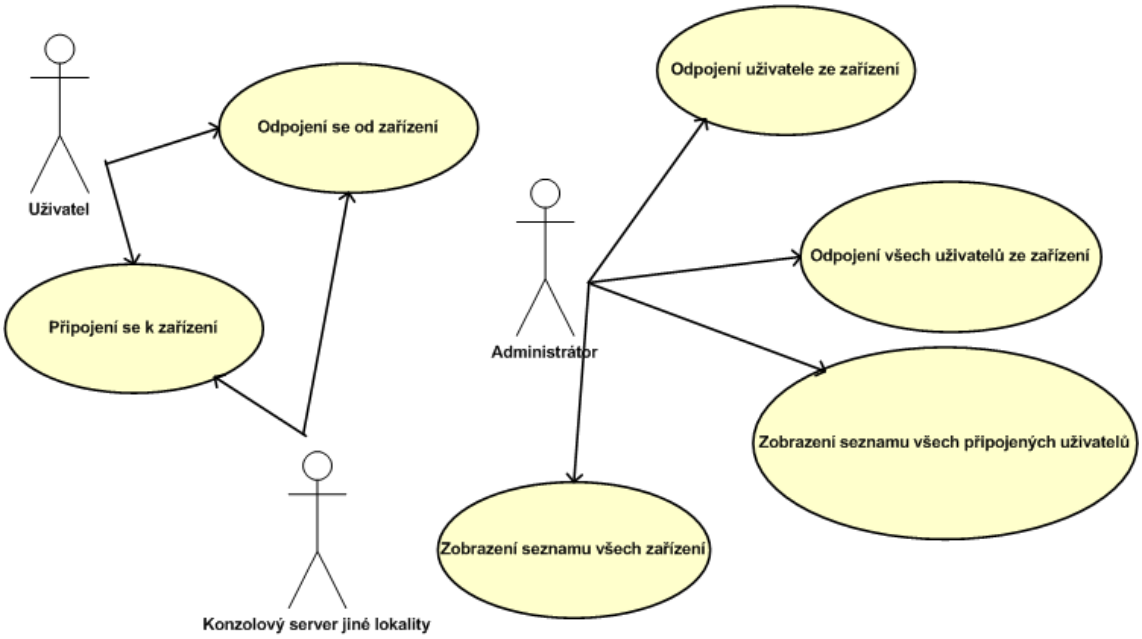
Konzolový server jiné lokality

Jako aktéra Konzolový server jiné lokality (dále jen Jiný server) bereme aplikaci Konzolového serveru, která se nachází v jiné lokalitě Virtuální laboratoře. Jiný server komunikuje s naším serverem, pokud se zařízení, na které se chce připojovat, nachází v naší lokalitě a Jiný server slouží pouze jako proxy server, který přeposílá veškeré požadavky našemu serveru. Jiný server není znovu autentizován a je na zařízení vpuštěn bez autentizace.

Administrátor

Jako aktéra Administrátora bereme správce lokality virtuální laboratoře, který má přístup k serverovým komponentám Virtlabu a jejich řídících rozhraních. Administrátor přistupuje ke Konzolovému serveru, pokud potřebuje odstranit uživatele ze zařízení, odstranit všechny uživatele ze zařízení anebo zobrazit seznam právě používaných zařízení.

Administrátor komunikuje s Konzolovým serverem skrze ovládací konzoli, kterou má každá serverová komponenta v sobě obsaženou.



Obrázek 17: USE-CASE diagram cílů jednotlivých aktérů

8.2 Případ užití č.1 - Připojení se k zařízení

Popis případu užití

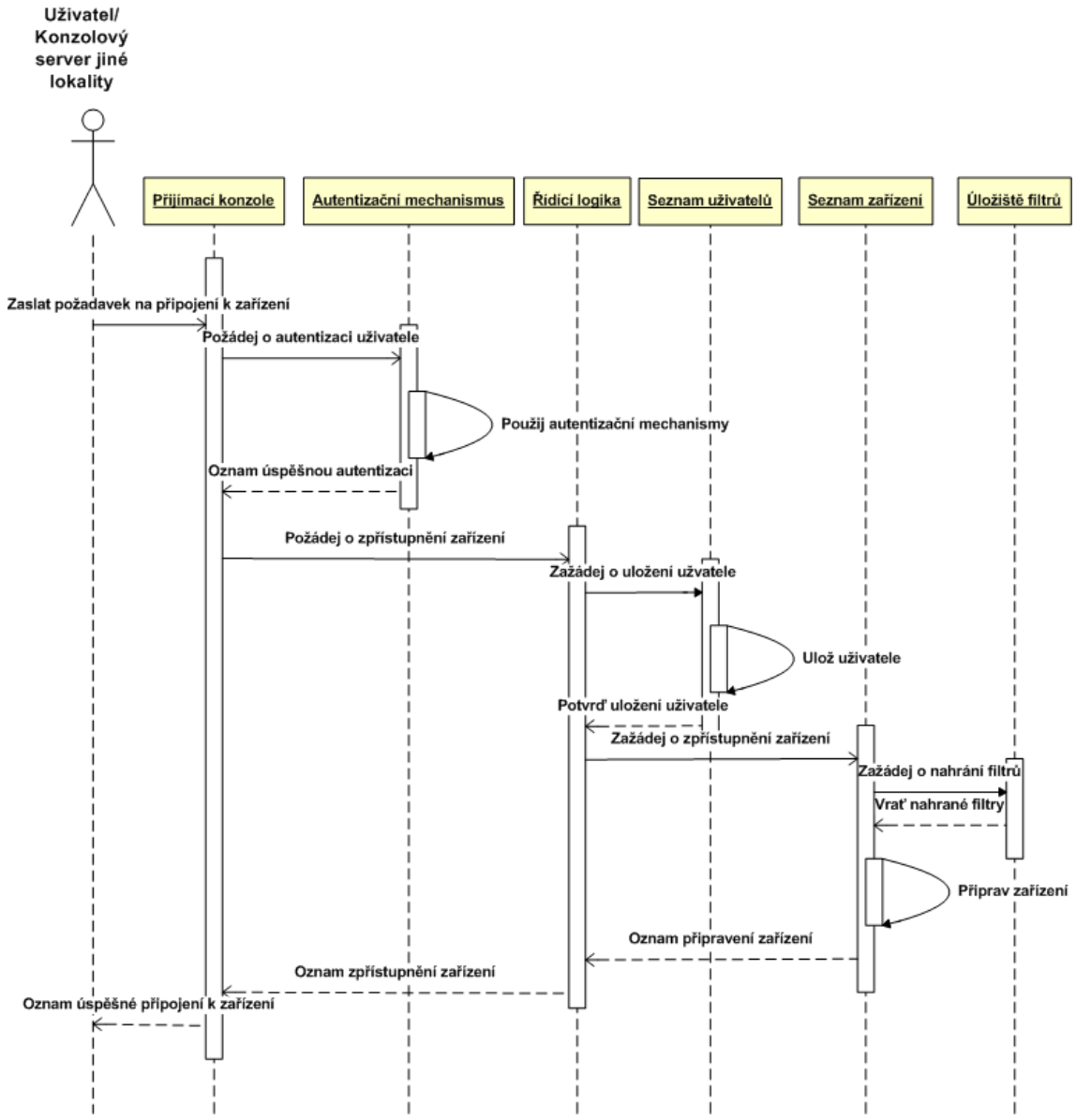
Uživatel nebo Konzolový server jiné lokality přistoupí na konzoli zařízení. Na Konzolovém serveru je, aby zjistil, zda je dané zařízení lokální a lze se na něj připojit anebo je zařízení z cizí lokality a je nutno přejít do proxy módu a předávat veškeré požadavky Konzolovému serveru jiné lokality. Dalším z úkolů je zjištění, ze které lokality uživatel pochází a zda je třeba ho autentizovat nebo ne. Posledním a nejdůležitějším úkolem je zpřiístupnění konzole uživateli a nastavení filtrů, které se na dané zařízení vztahují. Pokud je uživatel tutor, pak je ostataním uživatelům na konzoli zaslána notifikace o připojení tutora. Grafické znázornění základního toku lze vidět na obrázku 18.

Primární aktéři

- Uživatel
- Konzolový server jiné lokality

Základní tok událostí

1. Uživatel nebo Konzolový server jiné lokality se připojí na Konzolový server.
2. Konzolový server přijme Uživatele skrze daný Komunikační protokol.
3. Konzolový server autentizuje uživatele pomocí autentizačních metod, které má k dispozici.
4. Konzolový server zjistí, zda se zařízení nachází v dané lokalitě anebo ne.
5. Konzolový server přidá Uživatele do seznamu právě připojených uživatelů.
6. Konzolový server připraví veškeré filtry pro dané zařízení.
7. Konzolový server připraví spojení na zařízení.
8. Konzolový server přidá Uživatele ke konzoli zařízení a umožní mu práci se zařízením.
9. Pokud je uživatel tutor, pak Konzolový server upraví chování konzole zařízení podle módu tutora a notifikuje ostatní uživatele na konzoli o připojení tutora.



Obrázek 18: Sekvenční diagram základního toku Připojení se k zařízení

8.3 Případ užití č.2 - Odpojení se od zařízení

Popis případu užití

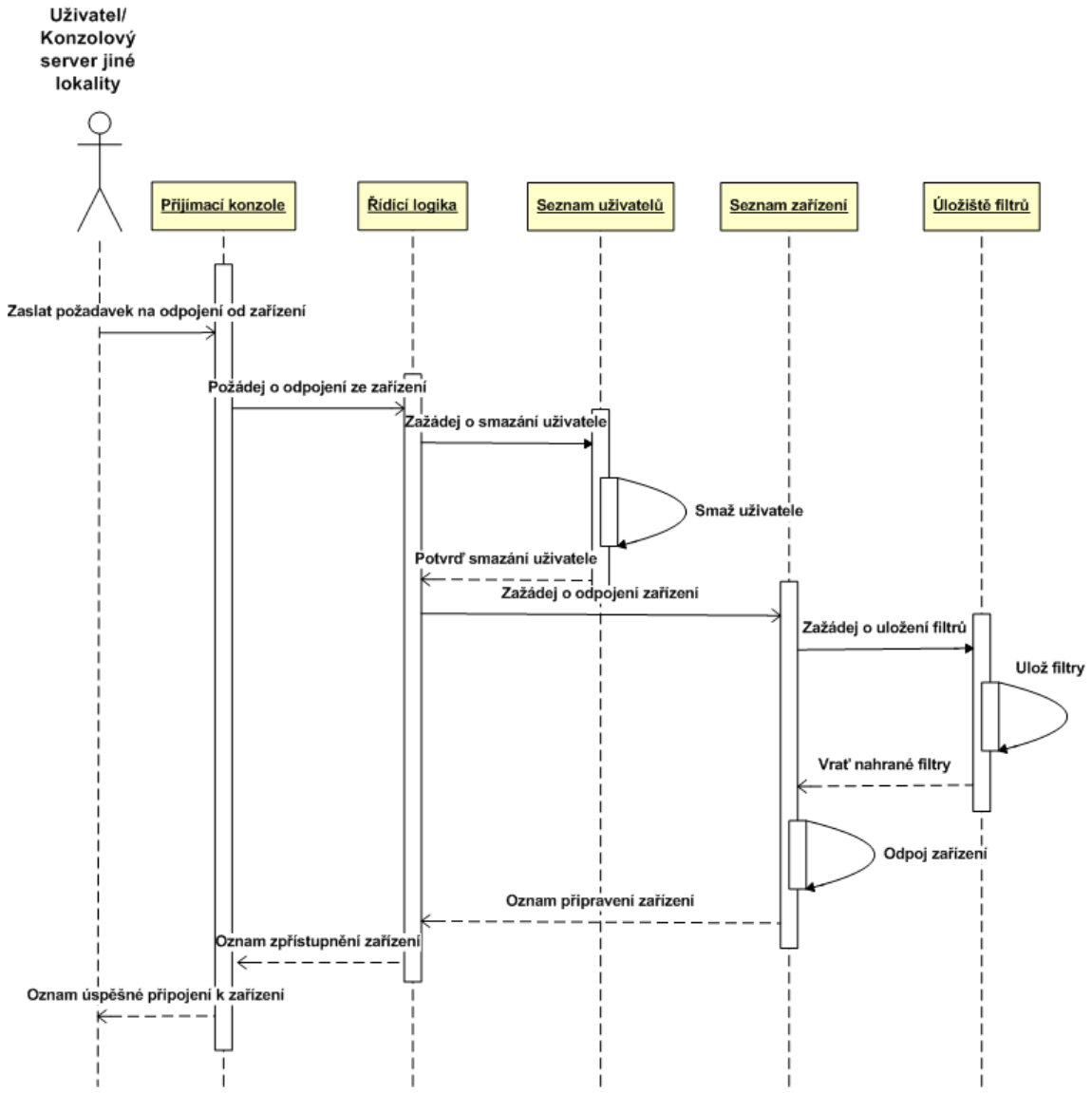
Uživatel nebo Konzolový server jiné lokality se odpojí z konzole zařízení. Na Konzolovém serveru je aby zajistil korektní odpojení uživatele od konzole, zjistil, zda se na konzoli zařízení ještě nacházejí uživatelé a popřípadě zajistit uložení stavu filtrů daného zařízení. Pokud se na konzoli zařízení nacházejí další uživatelé, pak pouze odpojí odhlašujícího se uživatele a neukládá stav filtrů. Pokud byl uživatel tutor, pak je ostatním uživatelům na konzoli zaslána notifikace o odpojení tutora. Grafické znázornění základního toku lze vidět na obrázku 19.

Primární aktéři

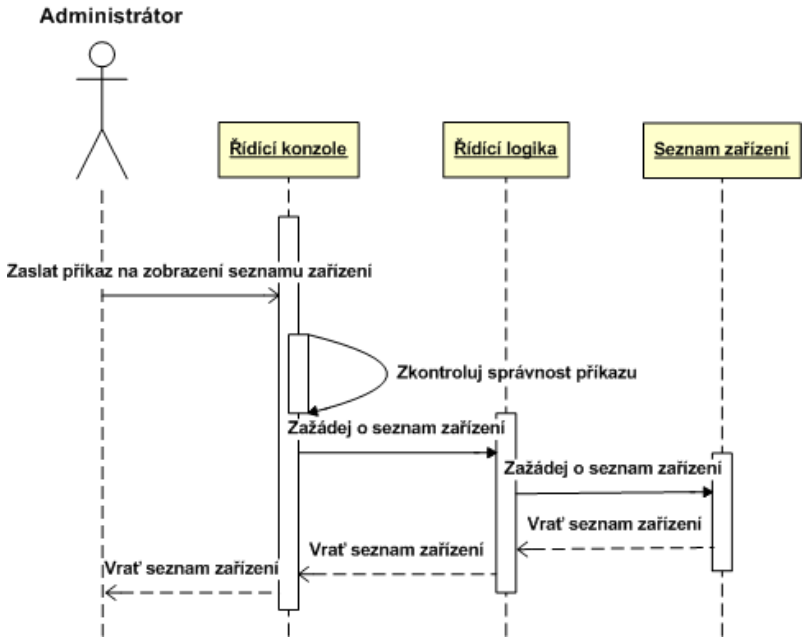
- Uživatel
- Konzolový server jiné lokality

Základní tok událostí

1. Uživatel nebo Konzolový server jiné lokality se odpojí od Konzolového serveru.
2. Konzolový server odpojí uživatele z konzole zařízení.
3. Konzolový server zkontroluje, kolik uživatelů se ještě nachází na konzoli zařízení, aby věděl, zda má již ukončit spojení na konzoli zařízení anebo ji má zachovat pro ostatní stále připojené uživatele.
4. Konzolový server se odpojí z konzole zařízení.
5. Konzolový server uloží stavy filtrů.
6. Konzolový server odpojí uživatele ze zařízení.
7. Pokud byl uživatel tutor, pak Konzolový server upraví chování konzole zařízení a nofitikuje ostatní uživatele na konzoli o odpojení tutora.



Obrázek 19: Sekvenční diagram základního toku Odpojení se od zařízení



Obrázek 20: Sekvenční diagram základního toku Zobrazení seznamu všech zařízení

8.4 Případ užití č.3 - Zobrazení seznamu všech zařízení

Popis případu užití

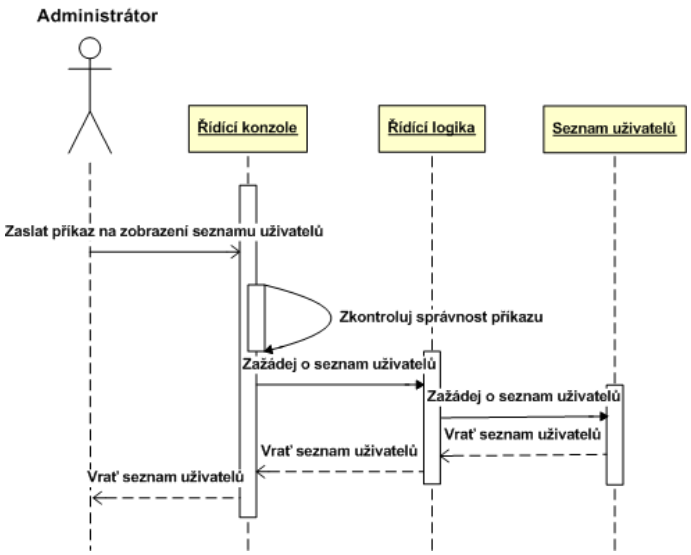
Administrátor zobrazí seznam všech zařízení, která jsou momentálně využita. Na Konzolovém serveru je, aby zkontroloval, zda je příkaz správně zadaný a má potřebné argumenty, vytvořil seznam všech právě používaných zařízení a tento seznam zpětně zobrazil administrátorovi. Grafické znázornění základního toku lze vidět na obrázku 20.

Primární aktéři

- Administrátor

Základní tok událostí

1. Administrátor zadá příkaz pro zobrazení právě používaných zařízení.
2. Konzolový server zkontroluje, zda je příkaz správně zadaný a má potřebné argumenty.
3. Konzolový server vytvoří seznam právě používaných zařízení.
4. Konzolový server zobrazí seznam zpět administrátorovi.



Obrázek 21: Sekvenční diagram základního toku Zobrazení seznamu všech uživatelů

8.5 Případ užití č.4 - Zobrazení seznamu všech připojených uživatelů

Popis případu užití

Administrátor se rozhodne zobrazit seznam všech uživatelů, kteří jsou momentálně připojeni na Konzolový server. Na konzolovém serveru je, aby zkontroloval, zda je příkaz správně zadaný a má potřebné argumenty, vytvořil seznam všech právě připojených uživatelů a tento seznam zpětně zobrazil administrátorovi. Zobrazeny jsou i proxy spojení. Grafické znázornění základního toku lze vidět na obrázku 21.

Primární aktéři

- Administrátor

Základní tok událostí

1. Administrátor zadá příkaz pro zobrazení právě připojených uživatelů.
2. Konzolový server zkontroluje, zda je příkaz správně zadaný a má potřebné argumenty.
3. Konzolový server vytvoří seznam právě připojených uživatelů.
4. Konzolový server zobrazí seznam zpět administrátorovi.

8.6 Případ užití č.5 - Odpojení všech uživatelů ze zařízení

Popis případu užití

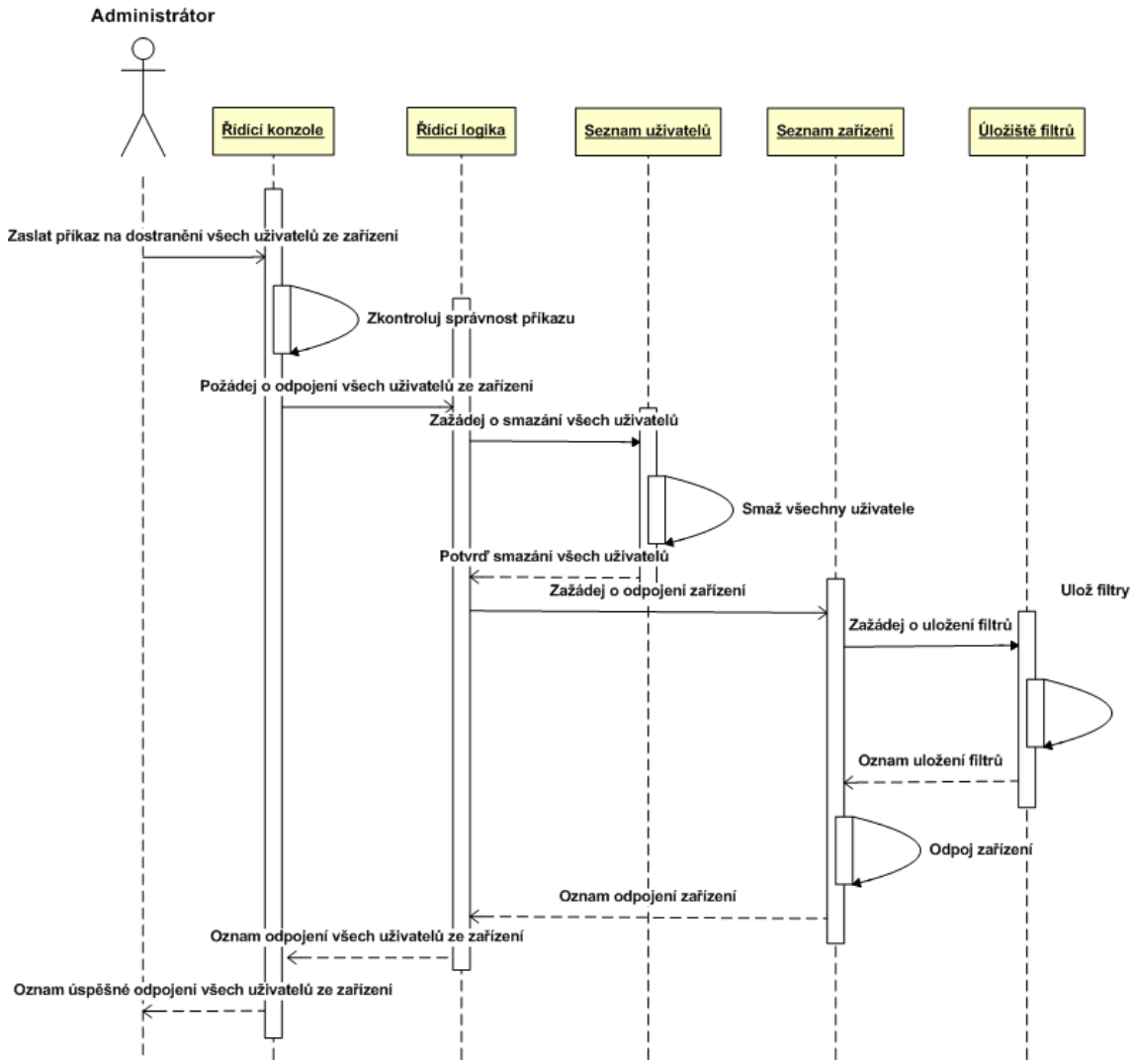
Administrátor se rozhodne odstranit všechny uživatele ze zařízení, které je momentálně využíváno Konzolovým serverem. Na Konzolovém serveru je, aby zkontroloval, zda je příkaz správně zadaný a má potřebné argumenty, odstranil všechny uživatele ze zařízení a poslal jim o tom zprávu, provedl korektní ukončení spojení na zařízení a v neposlední řadě informoval administrátora o úspěchu či neúspěchu. Grafické znázornění základního toku lze vidět na obrázku 22.

Primární aktéři

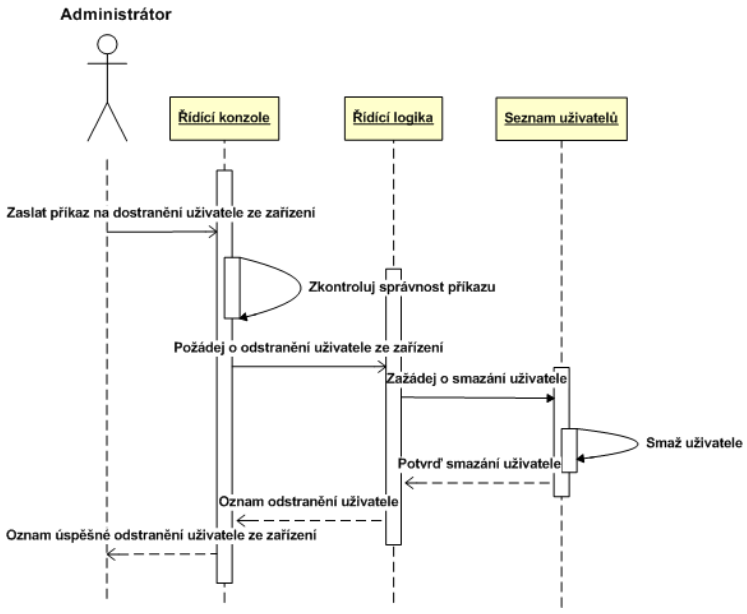
- Administrátor

Základní tok událostí

1. Administrátor zadá příkaz pro odstranění všech uživatelů ze zařízení.
2. Konzolový server zkontroluje, zda je příkaz správně zadaný a má potřebné argumenty.
3. Konzolový server odstraní všechny uživatele ze zařízení a informuje je o tom.
4. Konzolový server ukončí spojení na konzoli zařízení.
5. Konzolový server uloží stavy filtrů, protože již na konzoli není žádný další uživatel.
6. Konzolový server oznámí administrátorovi úspěšné provedení požadavku.



Obrázek 22: Sekvenční diagram základního toku Odstranění všech uživatelů ze zřízení



Obrázek 23: Sekvenční diagram základního toku Odstranění uživatele ze zařízení

8.7 Případ užití č.6 - Odpojení uživatele ze zařízení

Popis případu užití

Administrátor se rozhodne odstranit uživatele ze zařízení, které je momentálně využíváno Konzolovým serverem. Na Konzolovém serveru je, aby zkontroloval, zda je příkaz správně zadaný a má potřebné argumenty, odstranil uživatele ze zařízení a poslal mu o tom zprávu. Grafické znázornění základního toku lze vidět na obrázku 23.

Primární aktéři

- Administrátor

Základní tok událostí

1. Administrátor zadá příkaz pro odstranění uživatele ze zařízení.
2. Konzolový server zkontroluje, zda je příkaz správně zadaný a má potřebné argumenty.
3. Konzolový server odstraní uživatele ze zařízení a informuje jej o tom.
4. Pokud byl uživatel tutor, pak Konzolový server upraví chování konzole zařízení.
5. Konzolový server oznámí administrátorovi úspěšné provedení požadavku.

9 Konzolový server - Analýza a návrh

9.1 Architektura Konzolového serveru

Architekura Konzolového serveru, viz. obrázek 24, se skládá ze dvou komponent. Každá ze dvou komponent slouží k jinému účelu a dohromady zajišťují kompletní funkčnost Konzolového serveru. Tato rozdělí komponent si bere inspiraci z návrhového vzoru MVC. K této architektuře dále přináleží balíček, ve kterém se nacházejí třídy, které jsou využívány ve všech komponentách.

Komponenta číslo 1

Tato komponenta je jediná, se kterou uživatelé přímo komunikují. Stará se o příjem spojení uživatelů, zpracování Komunikačního protokolu, autentizaci a autorizaci uživatele a v neposlední řadě předává uživatelské spojení další komponentě ke zpracování. Dále se tato komponenta stará o příjem administrátorských připojení a zpracování administrátorských požadavků. Komponenta je reprezentována dvěma subsystémy, a to subsystémy, **Access_console_subsystem** a **Maitainance_subsystem**.

Komponenta číslo 2

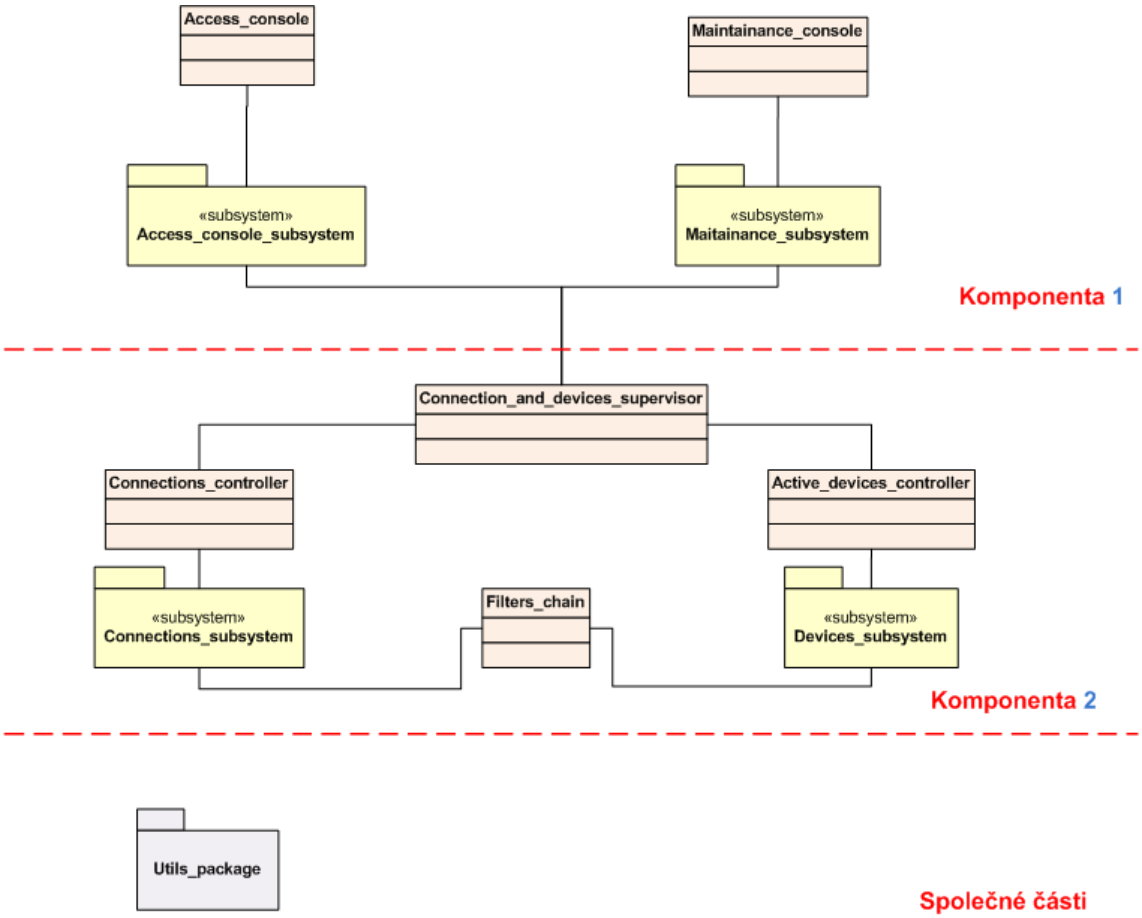
Tento komponenta se stará o zpřístupnění zařízení uživateli. Přijímá uživatelská spojení od první komponenty, k těmto spojením vytváří instance tříd, které reprezentují připojení k zařízení, načítá k těmto instancím množiny filtrů a v neposlední řadě zpřístupňuje uživatelům přístup na zařízení. Komponenta je reprezentována dvěma subsystémy **Connections_subsystem** a **Devices_subsystem**.

Společné části

Obě komponenty využívají balíček, který obsahuje třídy, které jsou využívány všemi předchozími komponentami. Obsahuje například třídy, které reprezentují sockety s rozšířenou funkcionalitou. Tato komponenta je reprezentována balíčkem **Utils_package**.

Třída: Connection_and_devices_supervisor

Jedná se o nejdůležitější třídu v celém Konzolovém serveru. Implementuje procedury, které se starají o přidávání uživatelských spojení, která jsou reprezentované třídou **User_connection**, odebírání uživatelských spojení, odpojování a připojování zařízení a pod. Kvůli paralelnímu charakteru Konzolového serveru se tato třída také stará o zamykání a serializaci požadavků na vykonání procedur. Třída využívá ke své práci instance dvou tříd a to tříd **Active_devices_controller** a **Connections_controller**, které využívá pro práci se zařízeními a uživatelskými spojeními. Třída taktéž obsahuje vlákno, které pravidelně kontroluje zda již nevypršel pro každé spojení čas rezervace a pokud ano, pak dané spojení odpojí od Konzolového serveru. Tato třída je spojnicí mezi komponentami a nenáleží tudíž do žádné z komponent.



Obrázek 24: Celkový pohled na architekturu Konzolového serveru

9.2 Subsystém Access_console_subsystem

Tento subsystém, viz. obrázek 25, se stará o příjem uživatelských spojení, zpracování komunikačního protokolu, autentizaci a autorizaci uživatelů a taktéž o vytváření instancí tříd, které reprezentují uživatelská spojení. Tato vytvořená spojení jsou pak předána instanci třídy **Connection_and_devices_supervisor** k dalšímu zpracování.

Třída: Access_console

Tato třída je jednou ze tříd, skrze kterou lze komunikovat s Konzolovým serverem. Jejím hlavním úkolem je přijímat uživatele skrze síťová spojení a vytvářet samostatná vlákna pro každého z uživatelů. Veškeré informace získané z připojení uživatele zaobalí do instance třídy **Client_initial_data**, kterou předá instanci třídy Komunikačního protokolu **Communication_protocol**. Výstup z Komunikačního protokolu, který tvoří instance třídy **Connection_info** předá instanci třídy **Authenticator_authorizator**. Pokud je klient autentizován a autorizován úspěšně, pak je vytvořeno objektové zaobalení uživatelského spojení, které je předáno instanci třídy **Connection_and_devices_supervisor** k dalšímu zpracování.

Třída: Communication_protocol

Jedná se o abstraktní třídu reprezentující Komunikační protokol mezi Konzolovým serverem a uživatelem. Dědičnost je využita z důvodů možné změny Komunikačního protokolu a je díky ní možné Komunikační protokol jednoduše vyměnit.

Třída: Communication_protocol_V1

Jedná se o třídu reprezentující konkrétně využívaný Komunikační protokol. Vstupem této třídy je instance třídy **Client_initial_data**, kterou Komunikační protokol využije ke komunikaci s uživatelem. Důležitou funkcí této třídy je parsování Komunikačního protokolu, validace dat a korektní Komunikace s jiným Konzolovým serverem, která se děje v případě využití proxy módu. Výstup z této třídy je instance třídy **Connection_info**, která obsahuje data přijatá skrze Komunikační protkol.

Třída: Client_initial_data

Tato třída obsahuje základní informace o uživateli jako je jeho IP adresa, port a ID vlákna, které uživatelské spojení obsluhuje.

Třída: Connection_info

Tato třída obsahuje veškeré informace, které Konzolový server získal od uživatele skrze Komunikační protokol. Obsahuje tedy ID rezervace, jméno zařízení a pod. Třída taktéž odkazuje na instanci třídy **Client_initial_data**, která obsahuje základní informace o uživateli.

Třída: Servers_controller

Tato třída obsahuje veškeré informace o všech ostatních Konzolových serverech. Zná adresy ostatních Konzolových serverů, jejich jména a taktéž obsahuje metodu, jejímž vstupem je adresa a výstupem je informace, zda se na této adrese nachází jeden ze známých Konzolových serverů. Známe Konzolové servery jsou získány z konfiguračního souboru. K využití této funkcionality potřebuje třída jako vstup instanci třídy **Connection_info**.

Třída: Authenticator_authorizator

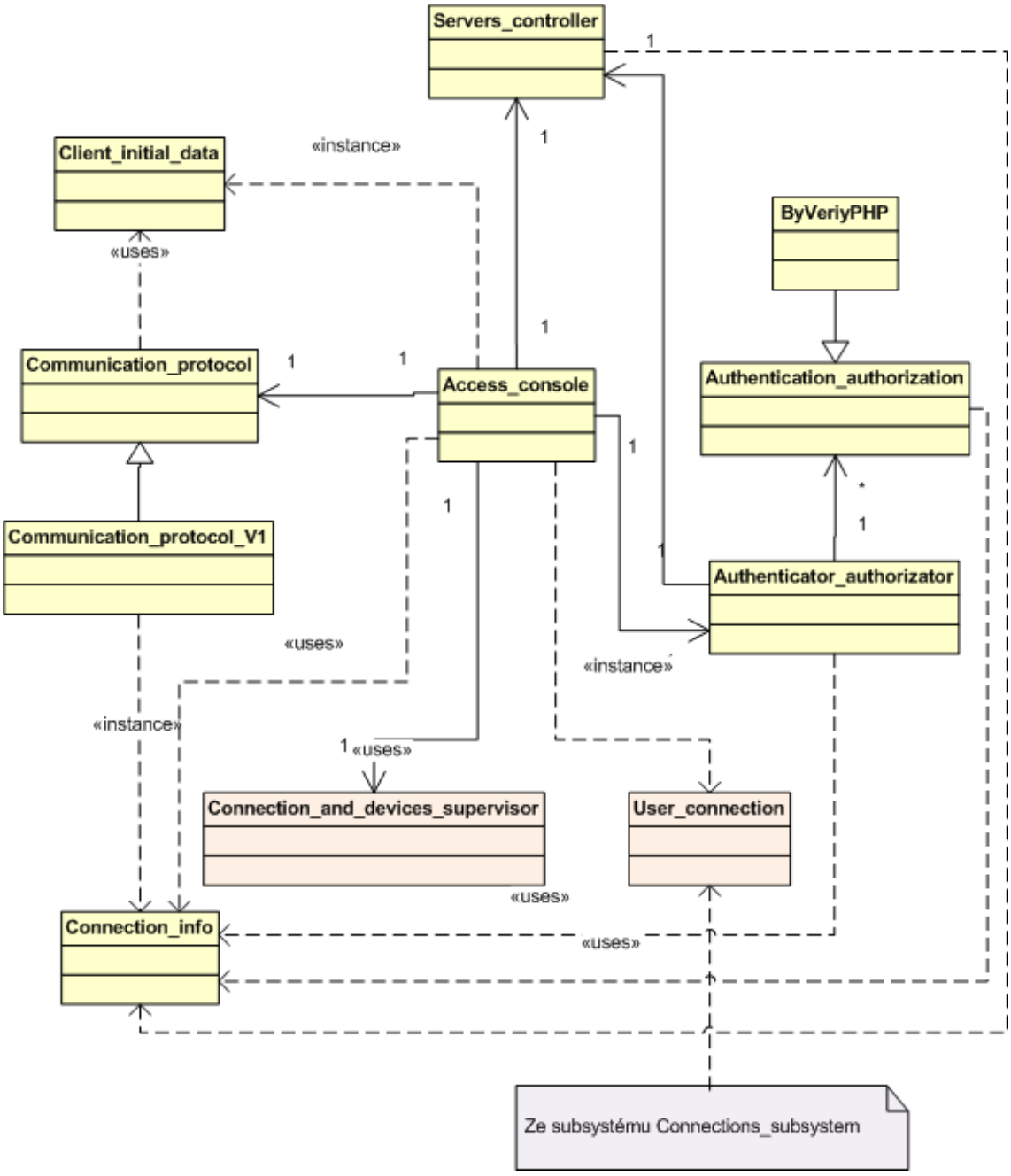
Tato třída se stará o autentizaci a autorizaci uživatele. Ke své práci využívá třída instanci třídy **Connection_info**, ze které čerpá informace o uživatelském spojení. Třída taktéž využívá ke své práci instanci třídy **Servers_controller**, ze které čerpá informace o původu uživatelského spojení. Pokud uživatelské spojení pochází ze známé lokality což se děje například při proxy módu, pak není potřeba autorizace a autentizace. K autorizování a autentizování uživatele používá třída instance třídy **Authentication_authorization**, kde každá z tříd reprezentuje jednu z možností autentizace a autorizace.

Třída: Authentication_authorization

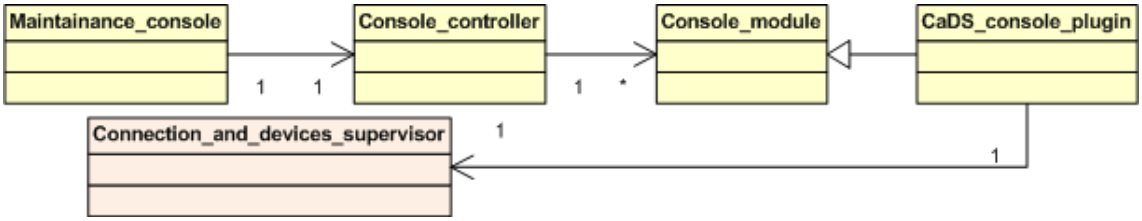
Tato abstraktní třída reprezentuje způsob autentizace a autorizace. Jako vstup využívá instanci třídy **Connection_info** a jako odpověď vrací, zda se autentizace a autorizace povedla nebo ne.

Třída: ByVerifyPHP

Tato třída reprezentuje momentálně jedinou používanou metodu autentizace a autorizace. Třída autentizuje a autorizuje uživatele oproti PHP skriptu verify.php, který oznamuje, kolik času zbývá do konce rezervace anebo záporné číslo, pokud není uživatel autentizován a autorizován. Ke své práci využívá třída socket s podporou SSL, skrze který komunikuje s řídícím serverem lokality.



Obrázek 25: Architektura subsystému Access_console_subsystem



Obrázek 26: Architektura subsystému Maitainance_subsystem

9.3 Subsystém Maitainance_subsystem

Tento subsystém, viz. obrázek 26, se stará o zpracování požadavků z administrátorské konzole. S tímto subsystémem komunikuje výhradně administrátor a upravuje pomocí něho chování Konzolového serveru.

Třída: Maintainance_console

Skrze tuto třídu jsou přijímány administrátorská spojení a pro každé je vytvořeno nové samostatné vlákno. Po přijetí nového spojení je administrátorovi zobrazena konzole se seznamem dostupných příkazů, který vzniká na základě komunikace s instancí třídy **Console_modules_controller**. Po přijetí příkazu z konzole je tento příkaz zaslán instanci třídy **Console_modules_controller** pro zpracování a vrácení odpovědi.

Třída: Console_modules_controller

Tato třída slouží ke zpracování požadavků přicházejících z instance třídy **Maintainance_console**. Jedná se o textové příkazy, které jsou zpracovány a následně jsou předány každému z modulů, které jsou reprezentovány instancí třídy **Console_module**. Pokud modul dokáže příkaz zpracovat, pak oznámí, že dokáže daný příkaz zpracovat. Tento modul je implementací návrhového vzoru *Chain-of-responsibility*. Kvůli možnosti mít připojeno více administrátorů, tato třída taktéž řeší serializaci příkazů.

Třída: Console_module

Tato abstraktní třída reprezentuje modul administrátorské konzole. Obsahuje metodu pro zpracování příkazů a zjištění informací o dostupných příkazech. Třída, která dědí z této třídy, může být později využita jako jeden z modulů administrátorské konzole.

Třída: CaDS_console_plugin

Tato třída dědí ze třídy **Console_module** a je tedy modulem administrátorské konzole. Zajišťuje administrátorovi možnost odpojení uživatele ze zařízení, odpojení zařízení, zobrazení seznamu všech uživatelů a zobrazení seznamu všech právě využívaných zařízení. Třída využívá ke všem úkonům třídu **Connection_and_devices_supervisor**

9.4 Subsystém Connections_subsystem

Tento subsystém, viz. obrázek 27, se stará o správu uživatelských spojení. Umožňuje přidávat uživatelská spojení, odebírat uživatelská spojení, zasílat uživatelům zprávy, vypisovat uživatelská spojení a pod. Subsystém je využíván instancí třídy **Connection_and_devices_supervisor**.

Třída: Connections_controller

Tato třída slouží jako správce a úložiště veškerých uživatelských spojení, která jsou momentálně připojena ke Konzolovému serveru. Stará se o ukládání uživatelských spojení, mazání uživatelských spojení, zasílání zpráv uživatelským spojením a pod. Každé z uživatelských spojení je reprezentováno instancí třídy **User_connection**.

Třída: User_connection

Tato abstraktní třída reprezentuje uživatelské spojení na Konzolový server. Enkapsuluje v sobě třídy **Client_initial_data** a **Connection_info**. Obsahuje tedy všechny potřebné informace o uživatelském spojení. Třída taktéž využívá instanci třídy **Filters_chain**, skrze kterou odesílá data zařízení. Referenci na tuto instanci získá třída až po připojení na zařízení.

Třída: Student_connection

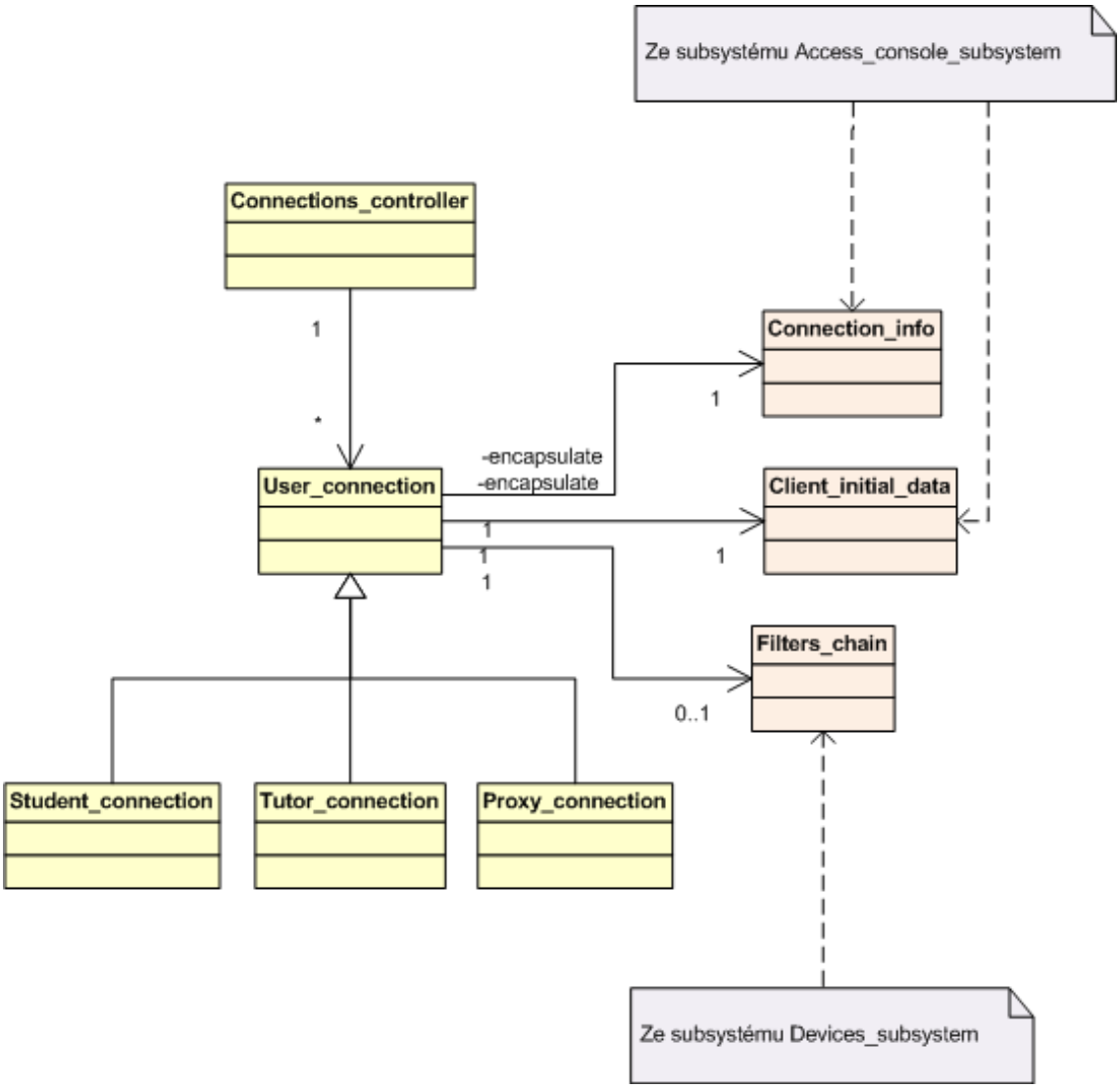
Tato třída reprezentuje studentské připojení ke Konzolovému serveru. Ke své práci využívá socket s nastaveným časovým limitem pro příjem dat. Třída implementuje chování zasílání dat na konzoli zařízení. Jelikož tutor ovlivňuje studenta, tak třída implementuje veškeré chování, které nastává po připojení tutora na konzoli zařízení.

Třída: Tutor_connection

Tato třída reprezentuje tutorské připojení ke Konzolovému serveru. Ke své práci využívá socket s nastaveným časovým limitem pro příjem dat. Třída implementuje chování zasílání dat na konzoli zařízení. Jelikož se tutoři mezi sebou neovlivňují, tak třída neimplementuje chování, které nastává po připojení tutora na konzoli zařízení, avšak informace o připojení tutora se objevuje i ostatním tutorům.

Třída: Proxy_connection

Tato třída reprezentuje proxy spojení na Konzolový server. Ke své práci využívá socket pro přeposílání dat. Jelikož je veškerá funkcionalita řešena až na cílovém Konzolovém serveru, tak se tato třída stará pouze o přeposílání dat mezi uživatelem a koncovým Konzolovým serverem.



Obrázek 27: Architektura subsystému Connections_subsystem

9.5 Devices_subsystem

Tento subsystém, viz. obrázek 28, se stará o správu zařízení. Stará se o zpřístupnění konzole zařízení, nahrávání filtrů, přidávání uživatelských spojení, vypisování aktivních zařízení a pod. Subsystém je využíván instancí třídy **Connection_and_devices_supervisor**.

Třída: Active_devices_controller

Tato třída slouží jako správce a úložiště veškerých aktivních zařízení. Stará se o vytváření připojení ke konzolím zařízení, které jsou reprezentovány instancemi třídy **Device_console** a o následné vytváření instancí třídy **Device**, která reprezentuje zařízení. K vytváření zařízení a připojení na konzoli zařízení využívá třída instance třídy **Device_info**, které obsahují všechny potřebné informace o zařízení. Třída taktéž používá instance třídy **User_connection** reprezentující uživatelská připojení a zpřístupňuje jim konzole zařízení.

Třída: Device

Tato třída reprezentuje zařízení. Slouží jako synchronizační bod mezi konzolí zařízení a uživatelským spojením. Jako synchronizační bod, myslíme místo na kterém se pomocí zamykání rozhodne, zda bude zapisovat jeden z klientů na zařízení anebo zařízení bude odpovídat všem klientům. Ke své práci využívá třídu **Device_console**, která reprezentuje připojení na konzoli zařízení a třídu **Filters_chain**, která reprezentuje seznam filtrů, skrze které musí projít uživatelská data než dorazí na konzoli zařízení. Třída taktéž využívá instanci třídy **User_connection**, kterou předá třída **Filters_chain** pro další zpracování.

Třída: Device_console

Tato abstraktní třída reprezentuje připojení na konzoli zařízení. Obsahuje metody pro odeslání dat na konzoli a pro příjem dat z konzole.

Třída: Telnet_console

Tato třída reprezentuje připojení na konzoli zařízení skrze telnetové připojení. Pro připojení k zařízení potřebuje znát IP adresu zařízení a TCP port, na kterém naslouchá konzole zařízení. Dědí ze třídy **Device_console** a je tedy jednou z možností připojení k zařízení.

Třída: Serial_console

Tato třída reprezentuje připojení na konzoli zařízení skrze sériové připojení. Pro připojení k zařízení potřebuje znát cestu k pseudosouboru ovladače sériového portu, na kterém naslouchá konzole zařízení. Dědí ze třídy **Device_console** a je tedy jednou z možností připojení k zařízení.

Třída: Filters_chain

Tato třída reprezentuje seznam filtrů, skrze který musí projít všechna data, která proudí mezi uživatelským spojením a konzolí zařízení. Třída využívá instance třídy **Filter**, které reprezentují jednotlivé filtry. Taktéž využívá instance třídy **User_connection** reprezentující uživatelská připojení, kterým odesílá data z konzole zařízení a přijímá od nich data pro konzoli zařízení. Dále využívá instanci třídy **Device**, které zasílá data od uživatelských spojení a přijímá od ní data pro uživatelská spojení. Veškerý provoz mezi oběma směry musí projít seznam filtrů. Třída dále využívá instanci třídy **Terminal_emulator**, kterou využívá pro zjištění momentálního zapsaného příkazu.

Třída: Filter

Tato abstraktní třída reprezentuje filtr provozu. Obsahuje metodu, která má za vstup data, která momentálně procházejí řetězem filtrů a jako odpověď vrací informaci, zda můžou daná data projít nebo jsou zahozena. Data se předávají po znacích.

Třída: Log_filter

Tato třída reprezentuje filtr, který slouží k ukládání všech příkazů, které byly zapsány na konzoli zařízení. Ke své práci využívá třídu **Terminal_emulator**, ze které zjistí, jaký příkaz byl zapsán a odeslán ke zpracování zařízení. Třída vždy propustí dál provoz mezi uživatelskými spojeními a konzolí zařízení a zapíše příkaz do logu.

Třída: Forbidden_commands_filter

Tato třída reprezentuje filtr, který slouží ke kontrole zakázaných příkazů. Tyto příkazy načítá jednorázově z databáze. Ke své práci využívá třídu **Terminal_emulator**, ze které zjistí, jaký příkaz byl zapsán a odeslán ke zpracování zařízení. Pokud se příkaz nachází mezi zakázanými příkazy, pak třída nepropustí dál provoz mezi uživatelskými spojeními a konzolí zařízení.

Třída: Terminal_emulator

Tato abstraktní třída slouží k emulaci terminálu a zjištění momentálního obsahu konzole. Obsahuje metodu pro vstup provozu mezi uživatelskými spojeními a konzolí zařízení a metodu pro výstup aktuálního obsahu konzole zařízení.

Třída: VT100_terminal_emulator

Tato třída reprezentuje emulaci terminálu využívající standrad VT100. Obsahuje metodu pro vstup provozu mezi uživatelskými spojeními a konzolí zařízení a metodu pro výstup aktuálního obsahu konzole zařízení. Tento emulátor momentálně pouze rozpoznává, jaký příkaz se nachází na konzoli zařízení a neinterpretuje veškeré příznaky terminálu VT100.

Třída: Devices_controller

Tato třída slouží jako správce zařízení, která nejsou momentálně využívána. Obsahuje informace o tom, zda je zařízení připojeno skrze sériový port nebo TCP spojení a taktéž seznam jejich filtrů. Třída využívá a vytváří instance třídy **Device_info**, které obsahují informace o zařízení a seznam jejich filtrů.

Třída: Device_info

Tato třída slouží k uchování informací o zařízení. Obsahuje informace o způsobu, jak se připojit ke konzoli zařízení a také seznam filtrů, které přináleží k danému zařízení.

Třída: Filters_controller

Tato třída slouží jako správce filtrů, které jsou využívány zařízeními. Obsahuje seznamy filtrů pro každé ze zařízení. Třída využívá a vytváří instance třídy **Filter**, které reprezentují jednotlivé filtry.

9.6 Balíček Utils_package

V tomto balíčku, viz. obrázek 29, se nacházejí třídy pro zajištění doplňkové funkčnosti. Najdeme zde například třídu pro práci se systémovými prostředky, třídu pro logování a pod.

Třída: Messages

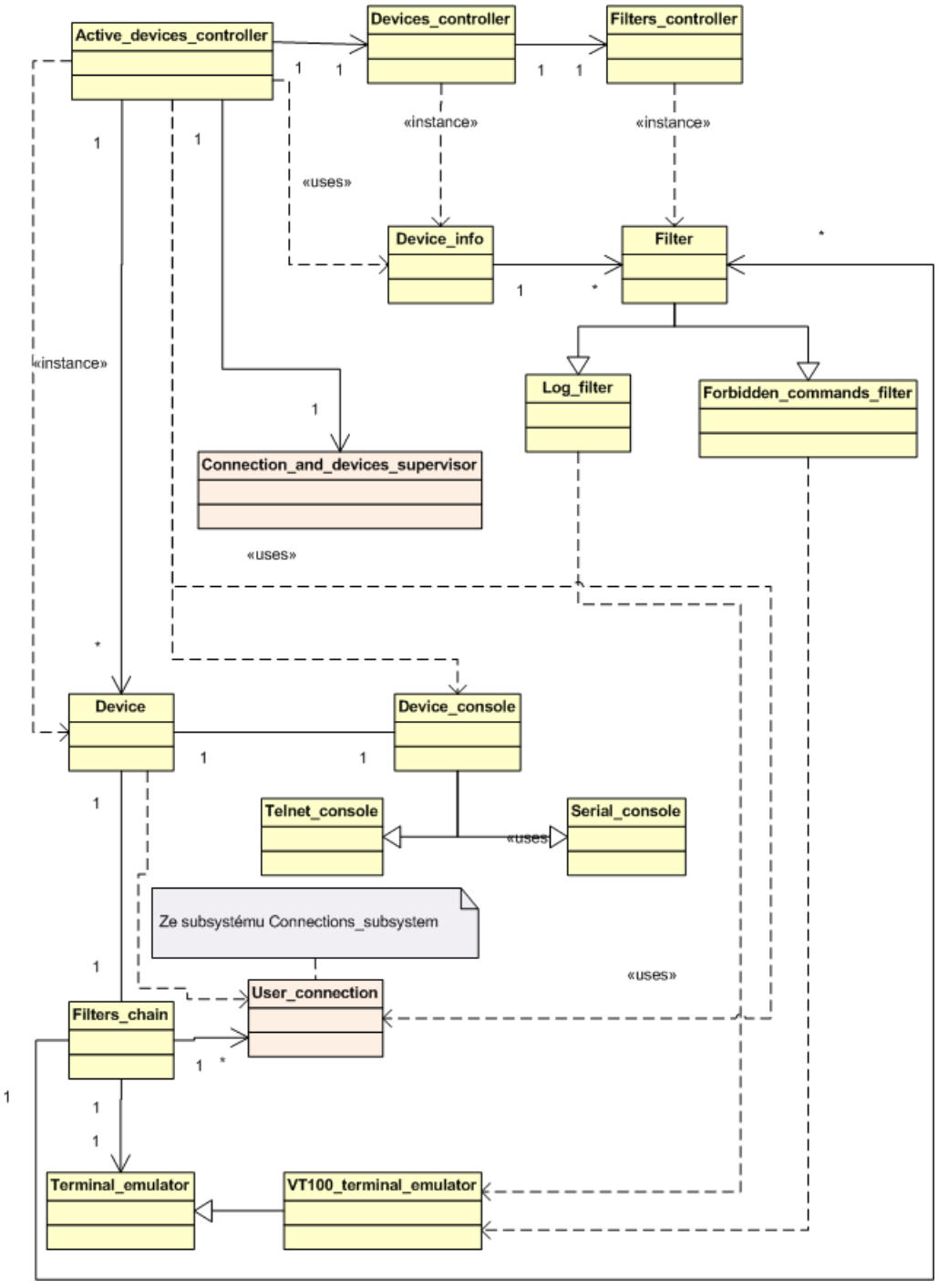
Jedná se o třídu sloužící k logování událostí do systému Syslog. Třída dodržuje logovací konvenci systému Virtlab, je globálně přístupná a může jí využít kterýkoliv modul, který má potřebu zalogovat některou z významných událostí.

Třída: Resources_controller

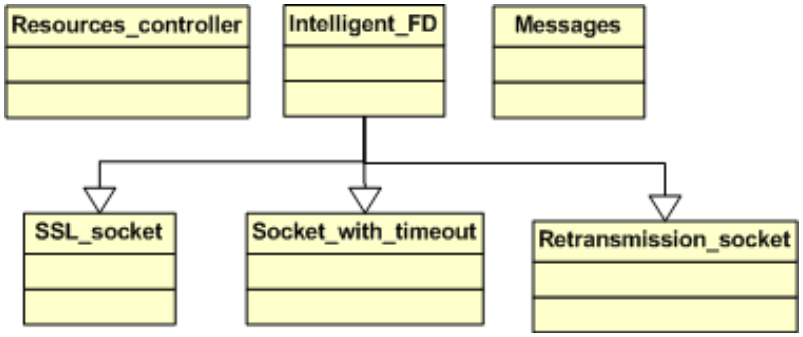
Tato třída se stará o správu dostupných zdrojů, které jsou přiděleny Konzolovému serveru. V současné době slouží k uchovávání počtu vláken. Jelikož jsou vlákna tvořena na více místech, je potřeba zaznamenat každé vytvořené nebo smazané vlákno a v případě dosažení maximálního počtu vláken, další vlákna nevytvářet. Hodnota by se dala přečíst i z OS, ale nenabízela by ochranu proti souběžným přístupům jako tato třída.

Třída: Intelligent_FD

Tato abstraktní třída reprezentuje množinu inteligentních FD, čímž myslíme FD, které obsahují přidanou funkcionalitu. Obsahuje návratové kódy a předpisy funkcí, které mají inteligentní FD používat.



Obrázek 28: Architektura subsystému Devices_subsystemm



Obrázek 29: Architektura balíčku Utils_package

Třída: Socket_with_timeout

Tato třída reprezentuje socket s možností nastavení vypršení časového limitu pro příjem dat. Tato třída je využíváná každým z modulů, který potřebuje komunikovat přes síť a zároveň mít možnost specifikovat, jak dlouho čekat na příjem dat.

Třída: SSL_socket

Tato třída reprezentuje socket s využitím SSL. Tato třída je využívána každým z modulů, které potřebují komunikovat pomocí SSL. K tomuto může dojít například při komunikaci s webovým serverem skrze zabezpečené připojení.

Třída: Retransmission_socket

Tato třída reprezentuje socket, který slouží k přeposílání dat mezi dvěma síťovými spojeními a slouží tedy jako proxy. Tato třída je využívána každým z modulů, které potřebují provozovat nerušenou výměnu dat mezi dvěma síťovými spojeními. Tato třída je například využita při proxy spojeních.

10 Konzolový server - Implementace

Konzolový server byl implementován s dodržením zásad Softwarového inženýrství. Blíže jsou zde popsány nejdůležitější problémy, které bylo nutné při implementaci vyřešit.

10.1 Práce s vlákny a zamykání

Veškeré zamykání a serializace se řeší ve třídě **Connection_and_devices_supervisor**. Před vykonáním každé z operací, které upravují nebo čtou ze seznamu připojených uživatelů nebo seznamu využívaných zařízení, se musí vlákno pokusit uzamknout synchornizační semafor. Pokud neuspěje, pak počká náhodnou dobu a vyzkouší to znovu. Jelikož v Konzolovém serveru existuje možnost, že může být vlákno zrušeno, v kteroukoliv chvíli je důležité definovat body, ve kterých může dojít k bezpečnému zrušení vlákna. Takovéto body jsou kdekoliv, kde se vlákno nenachází v kritické sekci. Proto před každým vstupem do kritické sekce znemožníme pomocí systémového volání ukončení vlákna a po opuštění kritické sekce systémovým voláním ukončení opět umožníme.

10.2 Přidání dalších filtrů

Pokud je potřeba přidat další filtry, pak je nutné postupovat pomocí následujícího postupu:

1. Dědit ze třídy **Filter**.
2. Přidat zpracování a vytvoření nového filtru do třídy **Filters_controller**.
3. Přidat název nového filtru do souboru s filtry pro zařízení.

10.3 Logování příkazů

Konzolový server loguje každý příkaz, který byl zapsán a potvrzen na konzoli zařízení. Stará se o to třída **Log_filter**. Každý příkaz je zapsán do log souboru a je k němu přidán čas potvrzení příkazu a ID uživatele, který příkaz potvrdil. Jednotlivé log soubory jsou pak řazeny do adresářů, které odpovídají ID rezervací a do souborů, které odpovídají jménu zařízení.

10.4 Oživení konzole

Po připojení k sériové konzoli zařízení Cisco často nastává situace, že konzole není přistupná a je potřeba jí oživit několika stisknutími klávesy ENTER. Jelikož přístupnou konzoli potřebují některé z dalších komponent Virtlabu bylo rozhodnuto, že o oživení konzole se postará Konzolový server. Oživení probíhá jednoduše. Nejdříve se pošle Cisco escape sekvence CTRL + R, která má za úkol zobrazit poslední použitý příkaz. Pokud na něj konzole nezareaguje, pak jsou celou minutu zasílány sekvence \r\n, dokud konzole nezareaguje. Po minutě je pokus o oživení ukončen a uživatel se musí pokusit připojit znovu.

10.5 Emulace terminálu

Emulace terminálu VT100 v současné implementaci interpretuje pouze několik sekvencí. Jsou to:

- Šipka nahoru, dolů, doleva a doprava
- Klávesy DELETE, ESCAPE, TAB, BACKSPACE, HOME, END, ENTER
- Sekvenci Ctrl + Z
- Vkládání znaků

Ostatní sekvence jsou ignorovány. I když se může zdát, že je tato množina sekvencí malá, stačí pro zjištění právě napsaného příkazu na konzoli. Pokud bude v budoucnu potřeba rozšířit nebo změnit emulaci terminálu, lze vycházet z této třídy.

10.6 Tutor mód

Tutor mód využijeme, pokud potřebujeme specifikovat jiné chování konzole zařízení. Módy tutora jsou následující:

- **off/shared** - Nepoužívá se žádný mód tutora. Konzole je sdílená mezi všemi uživateli.
- **observer** - Pozorovací mód. Tutor má možnost pouze pozorovat, co se děje na konzoli, ale nemůže do dění na konzoli zasahovat.
- **exclusive** - Exklusivní mód. Tutor zamkne celou konzoli pro sebe. Ostatní uživatelé nevidí a ani nemohou zasahovat do dění na konzoli.
- **master** - Master mod. Tutor zamkne konzoli pro sebe, ale ostatní uživatelé vidí, co se děje na konzoli.

11 Závěr

Závěrečná implementace Rezervačního serveru byla testována v období od 1.2.2010 do 14.2.2010. Testování zahrnovalo otestování nejdůležitějších funkcí jak v distribuovaném tak nedistribuovaném prostředí. K testování taktéž náležel integrační den, v jehož průběhu byl Rezervační server otestován s ostatními komponentami. V průběhu testování bylo nalezeno několik nesrovnalostí a chyb, které však byly opraveny. Na konci testování můžeme tedy prohlásit, že je Rezervační server plně funkční a je možné ho nasadit do produkčního prostředí Virtuální laboratoře.

Rezervační server byl nasazen do produkčního prostředí 15.1.2010. V průběhu nasazení byla přesunuta původní verze Rezervačního serveru do archívu a na její místo byla nasazena nová verze Rezervačního serveru. Nová verze byla nasazena na všech produkčních lokalitách a otestována. Nasazení proběhlo bez vážnějších komplikací.

Závěrečná implementace Konzolového serveru byla testována v období od 1.8.2009 do 1.9.2009. Testování zahrnovalo otestování nejdůležitějších funkcí jak v distribuovaném tak nedistribuovaném prostředí. K testování taktéž náležel integrační den v jehož průběhu byl Konzolový server otestován s ostatními komponentami. V průběhu testování bylo nalezeno několik nesrovnalostí a chyb které však byly opraveny. Na konci testování můžeme tedy prohlásit, že je Konzolový server plně funkční a je možné ho nasadit do produkčního prostředí Virtuální laboratoře.

Konzolový server byl nasazen do produkčního prostředí 15.9.2009. V průběhu nasazení byla přesunuta původní verze Konzolového serveru do archívu a na její místo byla nasazena nová verze Konzolového serveru. Nová verze byla nasazena na všech produkčních lokalitách a otestována. Nasazení proběhlo bez vážnějších komplikací. V průběhu nasazení se objevilo několik chyb, které se nepodařilo najít během testovací fáze. Veškeré chyby byly opraveny.

Na základě specifikace požadavků se podařilo vytvořit softwarová díla, která budou sloužit uživatelům distribuované Virtuální laboratoře. Obě díla byla vytvořena v daném termínu, byla otestována a nasazena do produkčního prostředí.

Obě díla byla vytvořena v souladu se zásadami Softwarového inženýrství [5] a byla k ním vytvořena kompletní dokumentace. Dokumentace bude sloužit dalším dalším vývojovým pracovníkům, kteří budou obě díla dále rozvíjet.

[5] Jednotlivé cykly a zásady vývoje jsou popsány v [10] a v [9]

12 Reference

[1] Pavel Herout, *Programovací jazyk C první a druhý díl*, Computer Press 2005

[2] Petr Grygárek, *Počítačové sítě*, VŠB - TUO Ostrava, http://www.cs.vsb.cz/grygarek/PS/index.html, poslední přístup 29.4.2010

[3] Petr Olivka, *Operační systémy*, VŠB - TUO Ostrava, http://poli.cs.vsb.cz/edu/osy/, poslední přístup 29.4.2010

[4] Petr Grygárek, *Terminologie distribuovaného Virtlabu*, VŠB - TUO Ostrava, 2007, http://www.cs.vsb.cz/vl-wiki/index.php/Virtlab:DistrTerminologie, poslední přístup 29.4.2010

[5] *Oficiální stránky projektu Virtlab*, http://www.virtlab.cz, poslední přístup 29.4.2010

[6] Richard Stones, Neil Matthew, *Linux - začínáme programovat*, Computer Press, 2000, ISBN: 80-7225-307-2

[7] Piotr Wroblewski, *Algoritmy, Datové struktury a programovací techniky*, Computer Press, 2004

[8] Rudolf Pecinovský, *Návrhové vzory*, Computer Press, 2007

[9] Joseph Schmuller, *Myslíme v jazyku UML*, Computer Press, 2001

[10] Ivo Vondrák, *Metody specifikace programových systémů*, Skripta VŠB - TUO, 2001

[11] Cockburn Alistair, *USE CASES - Jak efektivně modelovat aplikace*, Computer Press, 2005

[12] Stephen Prata, *Mistrovství v C++*, Computer Press, 2001