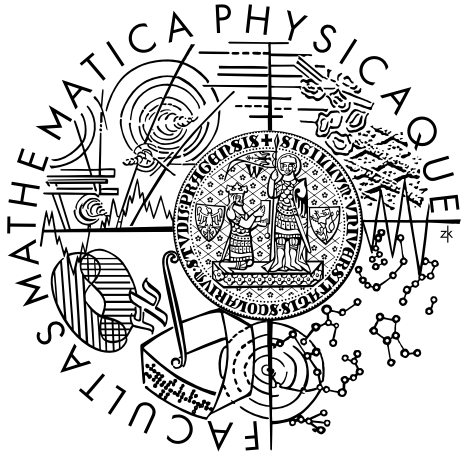
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Jan Drozen

Aplikace pro sledování a analýzu cévních výkonů

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Hoksza, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové systémy

Praha 2016

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 11.5.2016 Podpis autora

Název práce: Aplikace pro sledování a analýzu cévních výkonů

Autor: Bc. Jan Drozen

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Hoksza, Ph.D., Katedra softwarového inženýrství

Abstrakt: Jedním z oborů kardiovaskulární chirurgie je implantace a následná péče o vaskulární rekonstrukce a hemodialyzační přístupy. Vaskulární rekonstrukce, tzv. bypassy, i protetické hemodialyzační přístupy, tzv. grafty, se používají u pacientů, u kterých jiná léčba selhala. Jejich efektivita je tak zásadní pro život pacienta. Cílem této práce je návrh a implementace softwarového systému umožňujícího evidenci dat získaných při vaskulárních operacích a následných kontrolách. Důležitou částí systému je dále funkcionalita umožňující základní analytické a statistické zpracování evidovaných dat. Dalším cílem je z evidovaných dat získat nové znalosti použitelné pro zvýšení efektivity vaskulárních operací.

Klíčová slova: klient-server, WCF, datová analýza, databáze, cévní rekonstrukce, arteriovenózní graft

Title: Application for monitoring and analysis of vascular reconstructions

Author: Bc. Jan Drozen

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, Ph.D., Department of Software Engineering

Abstract: Implantation and the care of vascular reconstructions and accesses for haemodialysis is one of areas of cardiovascular surgery. Both vascular reconstructions, known as bypasses, and prosthetic haemodialytic accesses, known as grafts, are used only if conventional therapies are unsuccessful. Thus effectivity of these operations is critically important for patient survival. The aim of this thesis is development of software system for evidence data obtained during vascular operations and following checkups. Important is also support for basic analytical and statistical processing of collected data. Another goal is to transform collected data into knowledge useful for increasing effectivity of vascular operations.

Keywords: client-server, WCF, data analysis, database, vascular reconstruction, arteriovenous graft

Na tomto místě bych rád poděkoval mému vedoucímu, RNDr. Davidu Hokszovi, Ph.D., za vedení celé práce, mnoho cenných rad a usměrňování mých nápadů. Dále chci poděkovat MUDr. Róbertu Novotnému z II. chirurgické kliniky kardiovaskulární chirurgie Všeobecné fakultní nemocnice v Praze, který mne uvedl do tak zajímavé problematiky, jakou je cévní chirurgie. V neposlední řadě děkuji MUDr. Petru Vaverkovi, mému dobrému příteli, za odborné konzultace v oblasti medicíny, během kterých mi vysvětlil některé základní principy nezbytné k dalšímu studiu domény práce. A konečně děkuji rodině a všem, kteří mne při vývoji diplomové práce podporovali.

Obsah

Část I

Úvod a motivace

1. Úvod

Pokrok v medicíně za poslední století umožnil vznik nových léčebných postupů nebo celých nových odvětví. Jedním z relativně mladých oborů medicíny je i vaskulární chirurgie zabývající se různými onemocněními cév. Mezi pokročilé možnosti vaskulární chirurgie patří provádění cévních rekonstrukcí nebo implantace vaskulárních protéz. Pacienti podstoupivší cévní zákrok pak musí docházet na pravidelné kontroly, při kterých se zkoumá stav operačního pole a případně se provádí kroky nezbytné ke zlepšení stavu pacienta.

Ovšem ještě významnější pokrok zaznamenala informatika, a to ne za století, ale za posledních 50 let. Hlavně pokud pod pojmem „informatika“ myslíme obor v anglicky psané literatuře označovaný jako „computer science“. Zde můžeme vývoj dokonce kvantifikovat podle Mooreova zákona. Takto rychlý vývoj a navyšování výpočetního výkonu způsobilo pronikání informatiky do dalších vědních oborů, medicínu nevyjímaje.

Při kontrolách pacientů po vaskulárních rekonstrukcích jsou evidována data, jejichž následnou analýzou by mohly být nalezeny nové souvislosti, které by mohly přispět ke zefektivnění léčby. Například změnou volby materiálu pro rekonstrukci nebo úpravou medikace.

Byli jsme osloveni II. chirurgickou klinikou kardiovaskulární chirurgie Všeobecné fakultní nemocnice v Praze (VFN), abychom navrhli a implementovali softwarový systém pro evidenci a analýzu dat cévních rekonstrukcí. Údaje získané při kontrolách pacientů byly dosud evidovány pouze v papírových kartách, což zřejmě možnosti analýzy značně omezuje a software, který by splňoval zadané požadavky, dosud neexistoval. Naši mezioborovou práci tak řadíme do medicínské informatiky.

1.1 Struktura práce

Úvodní kapitolu věnujeme popisu domény rozdělené na cévní rekonstrukce a arteriovenózní grafty. Druhá část práce se zabývá vývojovou analýzou a implementací našeho softwarového systému. Postupně v jednotlivých kapitolách popíšeme databázovou, serverovou i klientskou část systému. Poslední kapitola v této části se věnuje možnostem rozšiřitelnosti systému. Rozšiřitelnost dále demonstrujeme na příkladech dvou konkrétních modulů. Závěrečná část práce obsahuje popis experimentů prováděných nad evidovanými daty.

2. Popis domény

Pokrok ve vědě a technice, který jsme zmínili v úvodu, s sebou přinesl i poměrně zásadní změny v životním stylu člověka. Tyto změny byly ve většině případů bezesporu pozitivní, avšak zásah do životního stylu tak složitého organismu, jakým člověk je, má i negativní dopady. Některé z těchto dopadů jsou známé pod názvem „civilizační choroby“. Mezi civilizační choroby řadíme podle [1] např.:

- kolorektální karcinom a další druhy rakoviny
- obezitu
- diabetes mellitus 2. typu
- ledvinové kameny
- osteoporózu
- Crohnovu chorobu a coeliakii
- kardiovaskulární choroby

Podle Světové zdravotnické organizace (WHO) jsou kardiovaskulární onemocnění vůbec nejčastější příčinou úmrtí na světě. Podle dat z roku 2012 na ně ročně zemře 17.5 milionu lidí, tj. tři lidé z deseti.

Z grafu 2.1 můžeme vyčíst, že první dvě položky, tedy ischemická choroba srdeční (ICHS) a cévní mozková příhoda (CMP), známá pod označením „mrtvice“, mají v součtu vyšší hodnotu, než je součet všech zbylých hodnot. Přitom právě ICHS a CMP jsou jediné dvě kardiovaskulární choroby v grafu.

2.1 Ateroskleróza

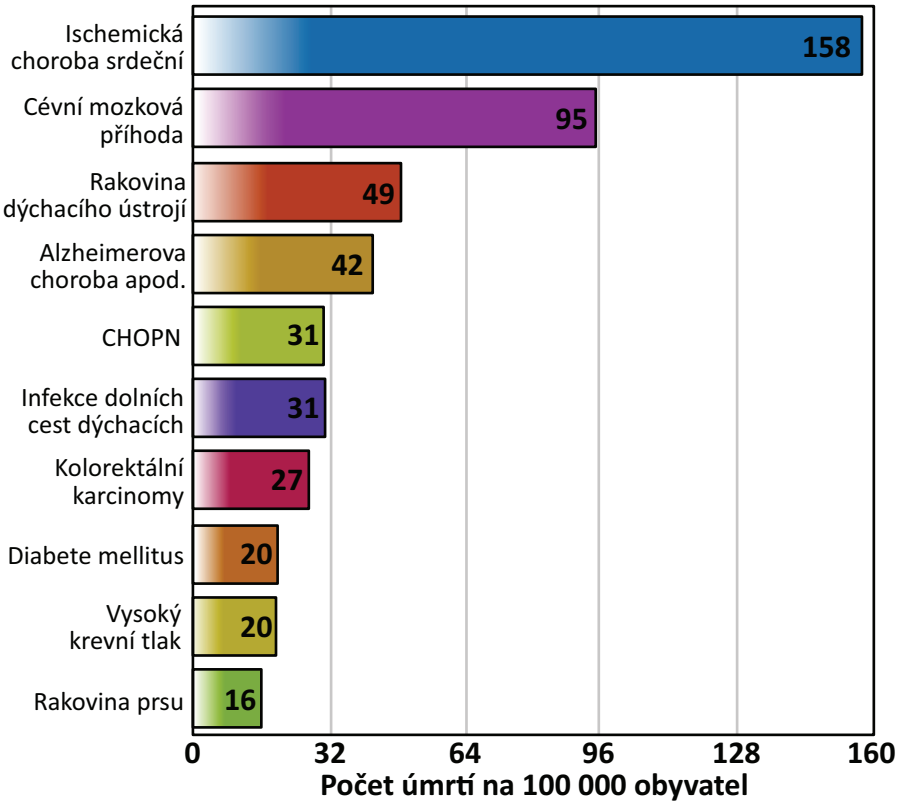
Nyní se zaměříme blíže na kardiovaskulární choroby. ICHS i CMP mohou progradovat až do iktu nebo infarktu s fatálními následky pro pacienta. Akutní stavy však bývají teprve důsledkem relativně dlouhodobého působení různých negativních faktorů.

Jedním z takových faktorů je arterioskleróza.

Definice 1 (Arterioskleróza)**.** *Arterioskleróza nastane, když cévy, které přivádějí kyslík a živiny ze srdce do zbytku těla tloustnou a tuhnou - což někdy omezuje průtok krve do orgánů a tkání. Zdravé tepny jsou pružné, ale v průběhu času mohou jejich stěny ztvrdnout. Tomuto procesu se obecně říká kornatění tepen.[2]*

Arteriosklerózu je možné dále specifikovat a jedním z jejich typů je tzv. **ateroskleróza**.

Definice 2 (Ateroskleróza)**.** *Ateroskleróza je kombinace změn intimy artérií sestávající z fokální akumulace lipidů, komplexních sacharidů, krve a krevních derivátů, fibrózní tkáně nebo kalcifikátů spojená se zdravotními změnami. [3, str. 4]*



Obrázek 2.1: Deset nejčastějších příčin úmrtí v zemích s vysokou ekonomickou úrovní v roce 2012. Zdroj: WHO. [2]

Uvedená definice, převzatá z [3], představuje starší pohled na aterosklerózu. Ten, jak je z definice patrné, považuje aterosklerózu za mechanický proces usazování lipidů a dalších látek na stěně artérie. Modernější přístupy však zdůrazňují důležitost zánětlivého procesu, resp. na aterosklerózu nahlížejí jako na „*imunitně zánětlivý (reparativní) proces, který je odpovědí na poškození intimy.*“[4, str. 60]. Úzká souvislost se zánětlivým procesem umožňuje lepší pochopení postupu aterosklerózy a souvislostí s rizikovými faktory[5]. Pro naši další práci však můžeme původní definici považovat za dostatečnou.

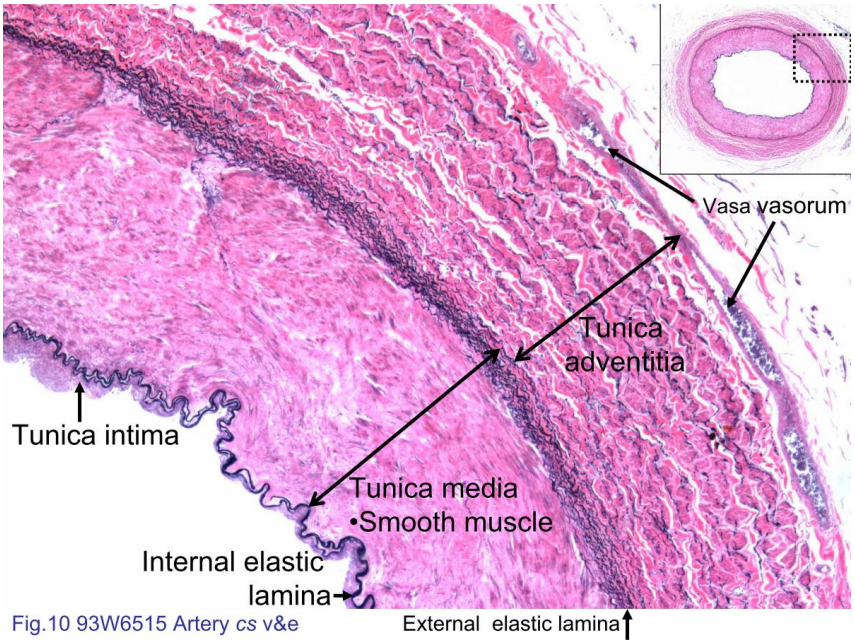
Přestože jsme se zmiňovali o civilizačních chorobách spojených s vývojem v relativně nedávné minulosti, ateroskleróza se v populaci vyskytuje již nepoměrně delší dobu[4]. Například u tzv. Ötziho[3] byla prokázána kalcifikace karotid.[4]

[2]The top 10 causes of death: `http://www.who.int/mediacentre/factsheets/fs310/en/` [citováno 11. května 2016]

[3]`https://cs.wikipedia.org/wiki/%C3%96tzi` [citováno 11. května 2016]

[4]World Heart Federation. "Otzi Iceman had genetic predisposition for atherosclerosis: Much the same in ancient peoples as it is today."ScienceDaily. ScienceDaily, 30 July 2014. `<www.sciencedaily.com/releases/2014/07/140730203707.htm>` [citováno 11. května 2016]

[4] `http://www.who.int/mediacentre/factsheets/fs310/en/index1.html` [citováno 11. května 2016]



Obrázek 2.2: *Řez tepnou obarvený Verhoeffovým barvivem pro zvýraznění elastických vláken a eosinem pro zvýraznění buněčných struktur. Svalové artérie mají více hladkého svalstva a méně elastinu v tunica media než elastické artérie. Svalové tepny jsou charakterizovány vrstvou vnitřní elastické membrány oddělující tunica intima od tunica media. Méně výrazná vrstva vnější elastické membrány leží mezi tunica media a tunica adventitia. Tunica intima sestává z endotelu a malého množství pojivové tkáně. Tunica adventitia se skládá z kolagenových vláken, elastických vláken a vasa vasorum.* [6]

2.1.1 Anatomie artérií

Abychom se mohli dále více věnovat nejen ateroskleróze, zmíníme se nyní krátce o anatomické stavbě artérií.

Artérie se skládá z různých vrstev (pro ilustraci viz obrázek 2.2 nebo 2.3), jejichž síla nebo význam se liší podle průměru a typu artérie[7]. Vrstvy jsou podle atlasu mikroskopické anatomie [6] v pořadí od nejsvrchnější tyto:

vasa vasorum
: doslova přeloženo jako „céva cév“, je vrstva kapilár a venul vyživujících tepny s velkým luminem

tunica externa
: nebo také **tunica adventitia** je vnější vazivové pouzdro

membrana elastica externa

tunica media
: vnitřní vrstva stěny složená z hladkých svalových buněk a elastických vláken

[6] Zdroj: Department of anatomy, Kaohsiung Medical University `http://anatomy.kmu.edu.tw/BlockHis/Block3/slides/block4_20.html` [citováno 11. května 2016]

[7] Existují anatomické odlišnosti, např. mezi cévami ve vysokotlakém řečišti (aortě) a řečišti s nízkým tlakem.

membrana elastica interna

tunica interna
nebo též **intima** je vnitřní vrstva tepny

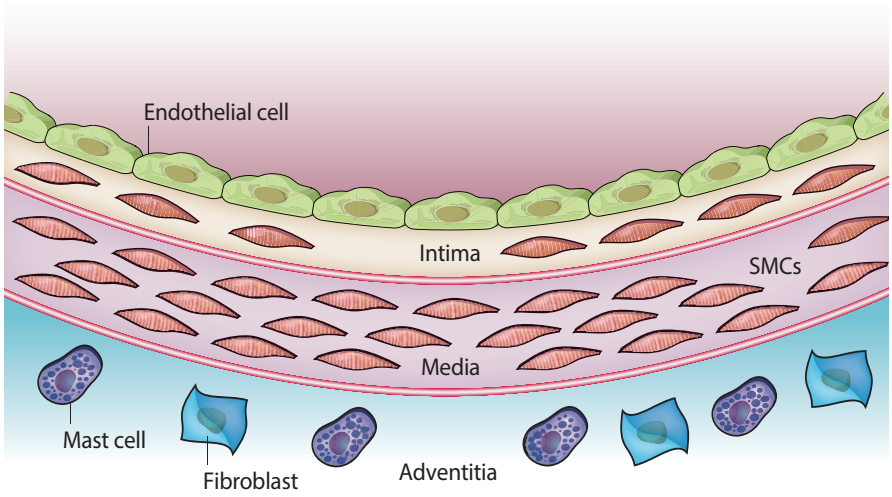
endotel
tenká zcela hladká vrstva umožňující plynulé proudění krve

lumen
vnitřní průměr (průsvit) tepny

Jak jsme uvedli v definici, ateroskleróza postihuje intimu, resp. endotel artérií, tedy dvě nejvnitřnější vrstvy stěny. Přestože je ateroskleróza, případně arterioskleróza obecně, považována za systémové onemocnění, její projevy, jak je opět zmíněno v definici, jsou lokální.

2.1.2 Lokální průběh aterosklerózy

Lokální projevy aterosklerózy se liší podle stupně rozvinutí choroby v daném místě. V průběhu aterosklerózy dochází na intimě ke vzniku lézí, kterým říkáme aterosklerotické pláty. Velikost i složení plátů závisí na stupni rozvinutí choroby. Jedná se tak o poměrně široký pojem zahrnující na jedné straně tenkou vrstvu lipidů a straně druhé i tromby znemožňující průchodnost cévy.

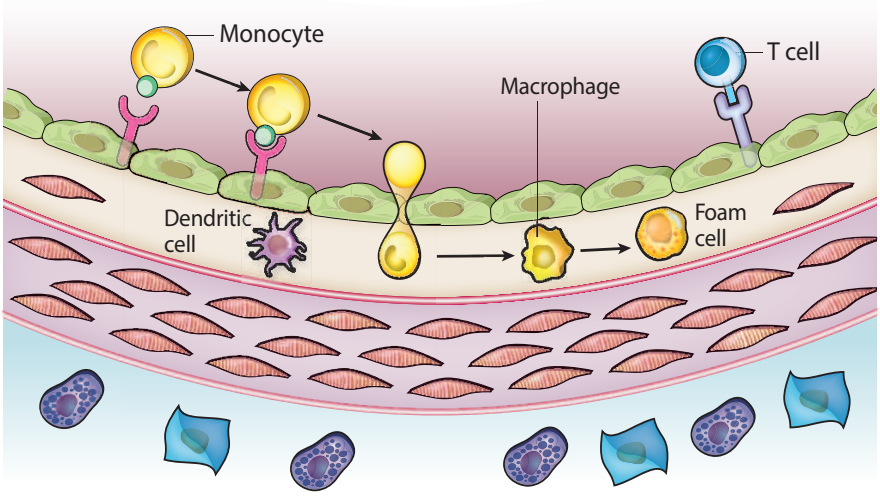


Obrázek 2.3: Schéma stěny artérie s rozlišením jednotlivých typů buněk v různých vrstvách. Převzato z [7].

.

1. fáze
Endotelové buňky, které zajišťují plynulé proudění krve, začínají na svém povrchu zachytávat leukocyty a zároveň se mění i propustnost endotelu. Ten pak umožňuje pronikání částic nízkodensitních lipoproteinů (LDL). Následně pak dochází k hromadění LDL v arteriální stěně. Monocyty, které se běžně vyskytují v intimě, diferencují v makrofágy, které dále fagocytují právě zmíněné částice LDL. To je jeden ze způsobů vzniku pěnité buňky

(FC) (viz obrázek 2.4). Druhý možný způsob vzniku pěnových buněk je z buněk hladké svaloviny pronikajících ze střední vrstvy (medie) do intimy a opět hromadění částic LDL [4].



Obrázek 2.4: Zachytávání leukocytů a vznik pěnových buněk tvořících tukové proužky. Převzato z [7].

2. fáze

Pro druhou fázi jsou typické tzv. tukové proužky. Tvoří je hlavně pěnové buňky prorůstající intimou a tvoří na ní žlutý povlak. Proužky neovlivňují lumen artérie a neomezují tak zásadně průtok krve. Tukové proužky se mohou vyvinout v další fáze aterosklerózy nebo naopak může dojít k regresi [4].

3. fáze

Pěnové buňky začínají odumírat a uvolňují malé množství extracelulárních lipidů. Vznikají tak **intermediární léze**.

4. fáze

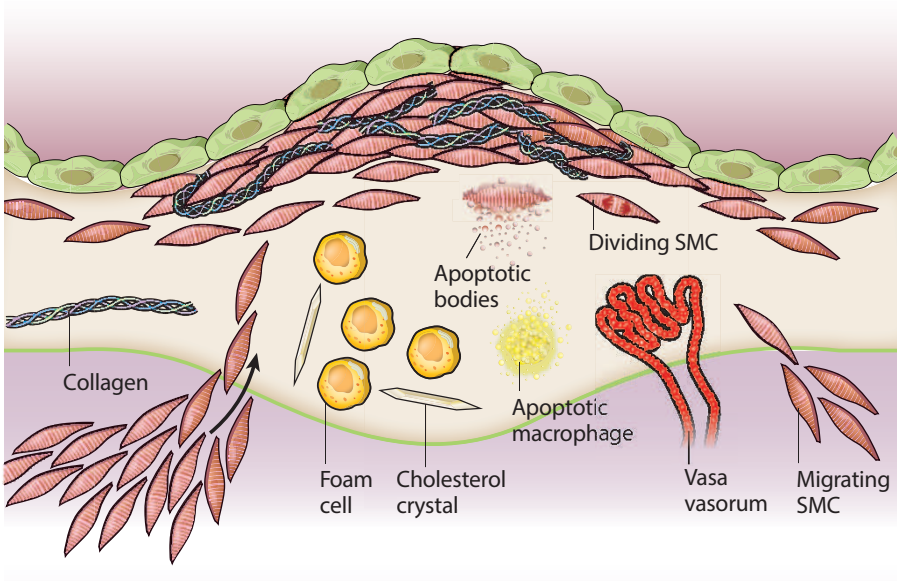
Do intimy dále pronikají buňky hladkého svalstva z medie a zároveň dochází k proliferaci původních buněk hladké svaloviny v intimě. Dále pokračuje ukládání extracelulárních lipidů a dalších látek tvořících extracelulární matrix (tvořenou zejména kolagenem). Tyto buňky a látky tvoří **ateromy** nebo též fibrózní pláty [4], které již zasahují do lumen.

5. fáze

Extracelulární lipidy tvoří lipidová jádra. Může docházet ke kalcifikaci. Aterom se tak mění ve **fibroaterom** (viz obrázek 2.5).

6. fáze

Posledním stádiem jsou **komplikované léze** vznikající masivní kalcifikací [4] fibroateromů. Může dojít k protržení fibrózní čepičky plátu, koagulaci, formování trombů a následné trombóze znemožňující průtok krve (viz obrázek 2.6).



Obrázek 2.5: Postupný vznik fibroateromu v 3. až 5. fázi aterosklerózy. Převzato z [7]

První fáze, kdy dochází ke změnám endotelu, se také nazýva *endotelová dysfunkce*.[8] Při endotelové dysfunkci jsou narušeny metabolické a sekreční vlastnosti endotelu [8]. Rozvoj endotelové dysfunkce, resp. aterosklerózy je ovlivněn různými tzv. rizikovými faktory (RF). RF dále dělíme na ovlivnitelné a neovlivnitelné podle toho, zda-li je může pacient ovlivnit svým chováním.

2.1.3 Neovlivnitelné rizikové faktory

Věk

Se vzrůstajícím věkem roste i riziko pokročilé aterosklerózy.

Pohlaví

Bylo prokázáno, že pro muže v produktivním věku je riziko aterosklerózy mnohem vyšší, než pro ženy. To je způsobeno ochrannými účinky estrogenu [4].

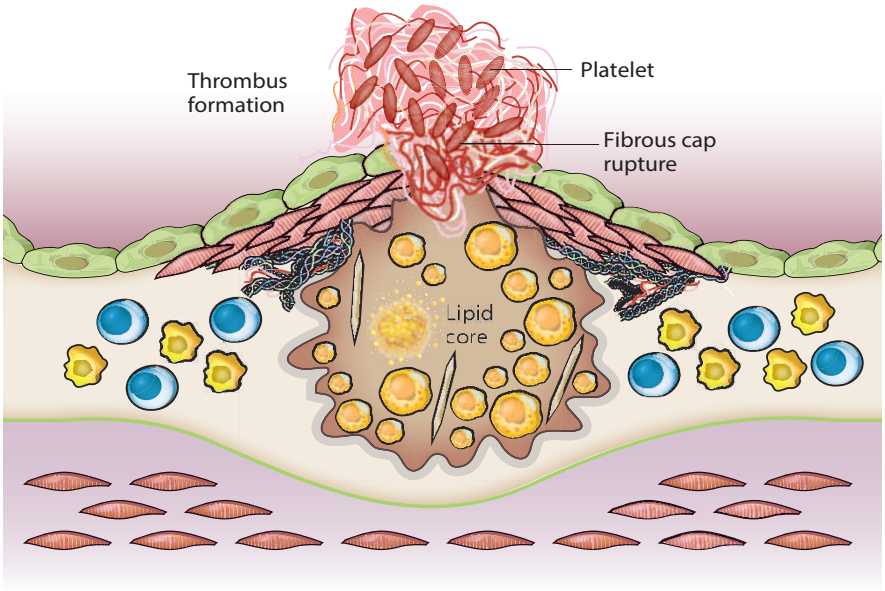
Genetické faktory

Riziko aterosklerózy také souvisí s rodinnou anamnézou. Zvláště důležitý je výskyt infarktu myokardu u přímého mužského příbuzného před 55. rokem [4][8].

2.1.4 Ovlivnitelné rizikové faktory

Ovlivnitelné RF jsou z našeho pohledu zajímavější. Zvláště pak v kontextu civilizačních chorob zmíněných na začátku kapitoly.

[8] Resp. endotelová dysfunkce je považována za 1. fázi aterosklerózy.



Obrázek 2.6: Ruptura komplikované léze, vznik trombu. Převzato z [7].

Kouření

Škodlivost kouření pro lidský organismus je dnes považována za fakt. V souvislosti s kouřením cigaret se hovoří hlavně o negativním účinku na plíce, ale účinek látek obsažených v cigaretovém kouři je také škodlivý pro cévní stěny. Kouření (i sekundární) ovlivňuje funkci leukocytů (imunosupresivní účinky), vasomotoriku a mění lipidový profil [9][10].

Diabetes mellitus II. typu

Diabetes mellitus (DM)[9] zvyšuje riziko aterosklerózy, stejně jako kouření, mnoha způsoby. Dochází ke změnám metabolismu a abnormální vasomotorice vedoucí k proliferaci buněk hladké svaloviny. Dále hyperglykémie zvyšuje oxidační stres, čímž přispívá k dyfunkci endotelu a inzulinová rezistence spolu s produkcí dalších látek napomáhá rozvoji aterosklerózy. [11][4]

Obezita

Obezita je jedním z rizikových faktorů kardiovaskulárních chorob obecně. V tomto kontextu je však spíše mezičlánkem k dalším rizikovým faktorům, jako je hypertenze nebo hladina LDL cholesterolu. Zajímavostí je, že první výzkumy odhalily pouze velmi slabou korelaci mezi obezitou a aterosklerózou [12].

Fyzická aktivita

Absence fyzické aktivity také patří mezi rizikové faktory. Možná bychom ji mohli zařadit i mezi civilizační choroby. Vyšší fyzická aktivita vede k lepší funkci endotelu a vasomotorice, čímž předchází rozvoji aterosklerózy. [13]

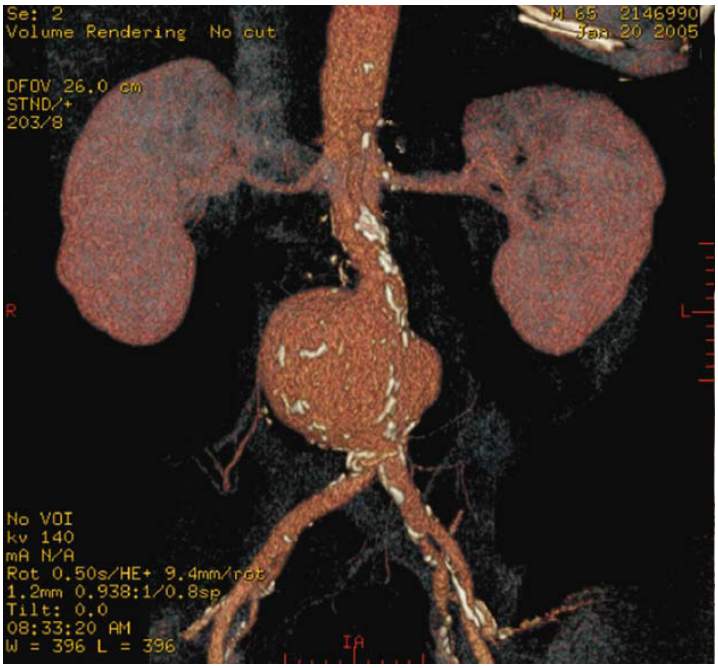
[9]Pokud budeme v dalším textu hovořit o diabetu, budeme myslet právě typ II, nebude-li explicitně uvedeno jinak.

Arteriální hypertenze

Arteriální hypertenze (AH), též vysoký krevní tlak[10], je obdobně jako DM jedním z rizikových faktorů kardiovaskulárních chorob. V reakci na AH se mění stěny postižených artérií a tím se mění i jejich vasomotorika. Působením vysokého tlaku může dojít k hyperplázii nebo hypertrofii hladkých svalových buněk v tunica media. Jak jsme již dříve zmínili, proliferace hladkých svalových buněk je jedním z kroků při rozvoji aterosklerózy. AH mění metabolismus elektrolytů a zvyšuje oxidační stres, což přispívá k dysfunkci endotelu.[14]

2.1.5 Aneurysma

Z hlediska průchodnosti artérie jsme o ateroskleróze doposud hovořili jako o procesu, který může způsobit trombózu, tedy že se lumen zužuje s rostoucím ateromem, na který posléze nasedá trombus. Může však docházet i k jiné závažné komplikaci, kterou je aneurysma nebo též výduť. Aterosklerotické pláty oslabují stěny artérií, které jsou pak křehčí a méně elastické. Pod vlivem tlaku krve, obzvláště, je-li tlak vysoký, pak může docházet k vytvoření výduti ve stěně artérie. Vznikne tak tzv. pravé aneurysma. Stěna pravého aneurysmatu se skládá ze stejných vrstev jako stěna artérie. Stěna aneurysmatu je však tenčí a se zvětšováním průměru výdutě roste i riziko ruptury.



Obrázek 2.7: CT snímek aneurysmatu abdominální aorty. Převzato z [15]

V souvislosti s aterosklerózou dochází nejčastěji ke vzniku aneurysmatu abdominální aorty (AAA). Ruptura AAA se vyznačuje vysokou celkovou mortalitou - podle některých zdrojů až 90%[16]. AAA nad bifurkací aorty zachycuje CT snímek na obrázku 2.7.

[10]Za hypertenzi se obvykle považuje tlak vyšší než 140/90 mm Hg. [8]

2.2 Ischemická choroba dolních končetin

Ateroskleróza typicky není diagnostikována jako primární onemocnění. Nicméně jiné choroby, které jsou více lokalizované, mohou mít s aterosklerózou, resp. jejím lokálním působením, přímou souvislost. Z našeho pohledu je nejzajímavější **ischemická choroba dolních končetin**. Jedná se, jak název napovídá, o stav, kdy dochází k nedostatečnému prokrvení dolní končetiny (viz CT snímek na obrázku 2.8).

Definice 3 (Ischemická choroba dolních končetin)**.** *Ischemická choroba dolních končetin (ICHDK) je způsobena zúžením nebo uzávěrem periferní tepny, nejčastěji na bázi aterosklerózy. Manifestace jsou od asymptomatických forem přes typické klaudikace po projevy kritické končetinové ischemie.[17, str. 3]*

Prof. Češka definici ještě rozšiřuje: *Kvalita života nemocných je omezena bolestí dolních končetin, rizikem vzniku tkáňových defektů a ztráty končetiny. Jedná se o onemocnění, které se vzhledem ke své aterosklerotické etiologii, při její časté generalizaci, vyznačuje vysokou morbiditou a mortalitou.*[4, str. 179]

Obrázek 2.8: CT snímek s obliterací pánevního řečiště. *Uzávěr AFS, difuzní stenotické postižení AP, uzávěr ATA, segmentální uzávěr AT. Vše na pravé dolní končetině.*[12]

V literatuře se uvádí, že až 90% případů ICHDK je způsobeno aterosklerózou[8]. Průběh ICHDK je možné rozdělit do několika stádií[13][4, str. 180]:

I. Asymptomatické stádium
Pacient nepociťuje žádné obtíže.

II. Klaudikační stádium

IIa. Nelimitující klaudikace
Klaudikace nad 200m chůze.

[12]Diagnostika dle MUDr. Róberta Novotného.
[13]Tzv. Fontainova klasifikace ICHDK.

IIb. Limitující klaudikace
Klaudikace pod 200m chůze.

III. Stádium klidové bolesti
Ischemická bolest postižené končetiny bez zátěže (obvykle v noci).

IV. Stádium kritické končetinové ischemie
Nehojící se defekty, nekrózy, gangrény.

2.2.1 Léčba ICHDK

V prvních dvou stádiích ICHDK je možná konzervativní léčba, tzn. úprava životosprávy (dieta, zákaz kouření, snížení tělesné hmotnosti) a medikace (nejčastěji antiagregancia na bázi kyseliny acetylsalicylové). V dalších fázích (IIb, III a IV) se přistupuje k invazivní terapii. Pro jednu z částí naší práce jsou stěžejní právě invazivní vaskulární terapie. Proto se v dalším textu zaměříme právě na tyto terapie a tak se ICHDK již nebudeme zabývat obšírněji.

2.3 Revaskularizační chirurgické a intervenční terapie

Společným cílem všech revaskularizačních metod je obnovení krevního oběhu v postižené části těla. Různé typy postižení se pak řeší pomocí různých metod. Volba metody závisí mimo jiné na:

- typu postižení (např. uzávěr, aneurysma)
- umístění (průměr lumen artérie, typ artérie, přístupnost)
- stav řečiště v okolí postiženého místa

Z našeho pohledu můžeme cévní zákroky rozdělit do dvou skupin:

1. **endovaskulární intervence**
2. **chirurgické terapie**

Endovaskulární intervence používají pro přístup k postiženému místu cévní řečiště. Typicky se na nějakém dobře přístupném místě (nejčastěji do femorální artérie [18]) zavede katetr, který se protáhne až do místa postižení. Typ a hlavně konec katetru pak odpovídají příslušnému typu zákroku.

Oproti tomu chirurgické terapie používají přímý (chirurgický) přístup k postiženému místu.

Pro ilustraci uveďme příklady možné desobliterace - odstranění překážky bránící průtoku artérií (základní chirurgické cévní výkony[18]):

trombembolektomie
Odstranění embolie nebo trombů uzavírajících lumen. Možné je řešení chirurgické i endovaskulární. Chirurgické řešení (tzv. přímá trombembolektomie) znamená vyjmutí embolu přímo z otevřené artérie. Endovaskulární řešení (tzv. nepřímá trombembolektomie) využívá balónkové nebo spirálové katetry, kdy dojde k vytlačení nebo vytažení embolu z postiženého místa.

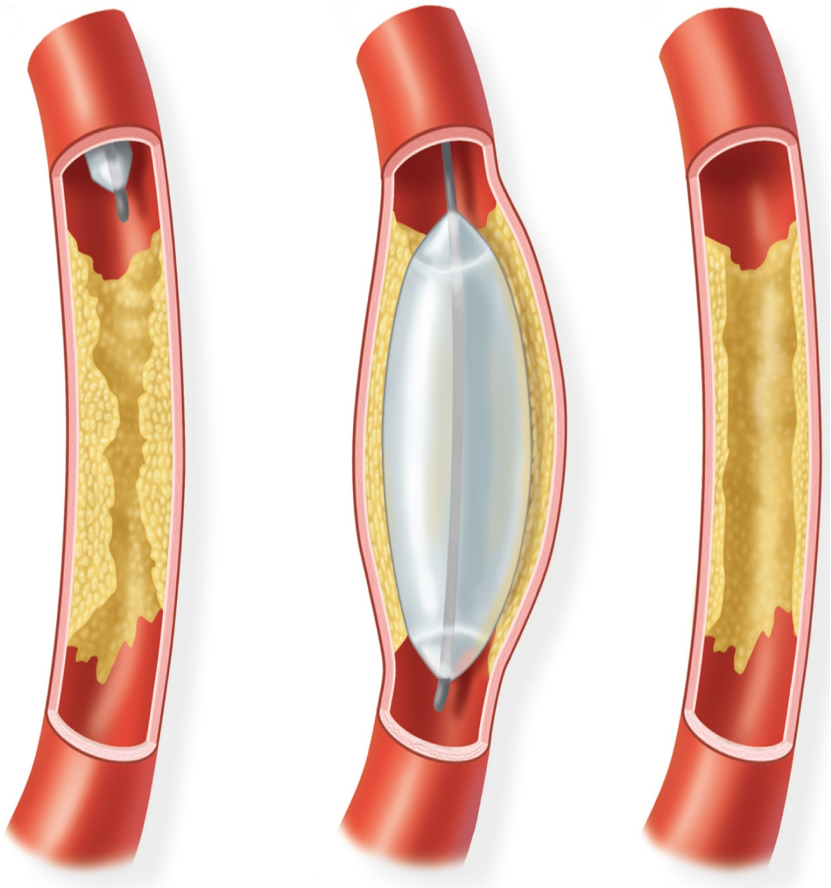
endarterektomie

Při endarterektomii dochází k odstranění aterosklerotického plátu ze stěny postižené artérie. Používá se chirurgický přístup, kdy se plát odstraňuje přímo z postiženého místa a příslušné vrstvy cévní stěny.

Nyní se blíže zaměříme na tři konkrétní typy cévních zákroků.

2.3.1 Perkutánní transluminální angioplastika, stent

Perkutánní transluminární angioplastika (PTA) je, jak už název napovídá, endovaskulární zákrok, který se využívá ke zlepšení průtoku krve při stenóze cévy. Používá se balónkový katetr[14], který se zavádí přímo na postižené místo. Balónek se pak nafoukne (nebo naplní fyziologickým roztokem), čímž dojde k mechanické dilataci lumen. Roztažením balónku dojde k porušení struktury stěny cévy.



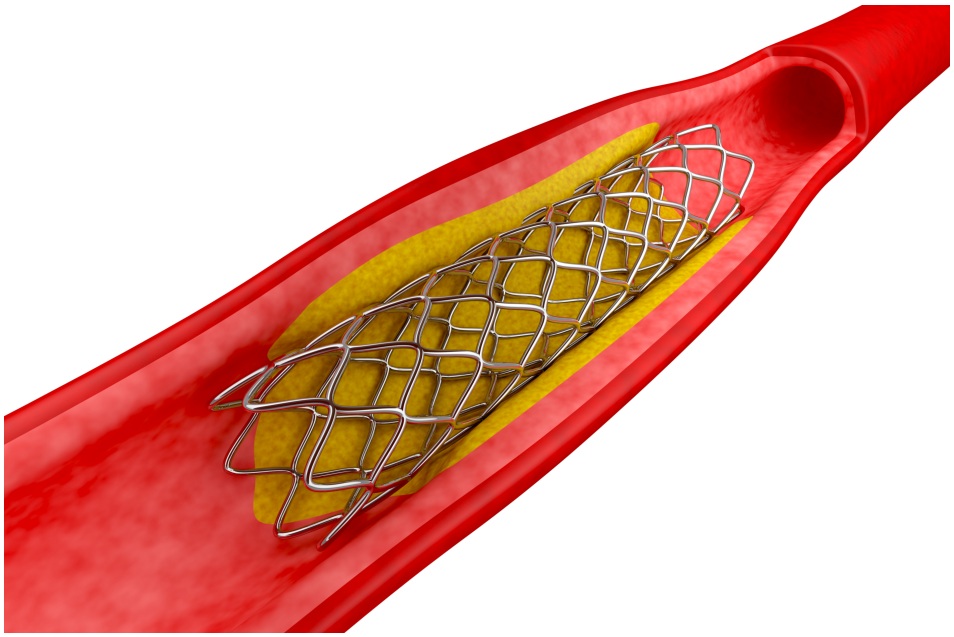
Obrázek 2.9: Průběh PTA[15].

Toto porušení, v podstatě roztržení povrchu stenotické léze (aterosklerotického plátu), hladkých svalových buněk a kolagenu, je žádoucí a podílí se na výsledku angioplastiky. V ideálním případě je dilatace trvalá. Pro výsledek PTA hraje zásadní roli složení stenotické (aterosklerotické) léze[19].

V některých případech může být PTA nedostatečná, kdy po vytažení katetru nedojde k obnovení dostatečného průměru artérie. V takovýchto případech je navíc možné zavést tzv. stent. *„Stenty jsou endovaskulární výztuže, které mají*

[14]Katetr, který má na konci balónek.

pomocí své radiální síly přemoci kompresivní sílu stenotické léze a pomocí kruhové pevnosti odolat zevní kompresi.“ [18, str. 16]



Obrázek 2.10: Umístění stentu po PTA. [16]

2.3.2 Cévní rekonstrukce, bypass

V případě, že nelze provést endovaskulární intervenci, nebo pokud jsou intervence neúspěšné, je možné přistoupit k chirurgické terapii. Přirozenými dvěma možnostmi je postižené (uzavřené) místo artérie obejít, nebo jej nahradit.

V obou případech jde o tzv. bypass neboli cévní rekonstrukci. Krev je z artérie odváděna typicky proximálně od uzávěru a po průtoku bypassem navrácena do artérie distálně od uzávěru, čímž je obnoveno krevní zásobení postižené oblasti.

Bypassy je možné klasifikovat podle jejich vedení:

anatomicky
: Anatomicky uložený bypass je veden podél rekonstruované artérie.

extraanatomicky
: Extraanatomický bypass je vedený mezi dvěma různými artériemi.

Bypass je na cévu možné napojit (našít) dvěma způsoby. První možností je našití bypassu na cévní stěnu, čímž vznikne tzv. *end-to-side* anastomóza. Druhou možností je spojení, při kterém lumen bypassu navazuje na lumen cévy. V tomto případě se jedná o anastomózu *end-to-end*. V určitých případech může být volba typu anastomózy determinovaná typem rekonstrukce nebo jinými okolnostmi. V případě end-to-side anastomózy může být příkladem nutnost zachování oběhu distálně od anastomózy. Naopak end-to-end anastomóza může být nezbytná v situaci, kdy musí být poškozená část cévy odstraněna. Závislost typu anastomózy

[16] Zdroj:Union Memorial Hospital Stent Lawsuit `http://bit.do/unionmemorialstent` [citováno 11. května 2016]

(typicky proximální) a pooperačního vývoje je předmětem výzkumu. Výsledky ukazují, že v dlouhodobém pooperačním vývoji nejsou zásadní rozdíly [20] [21].

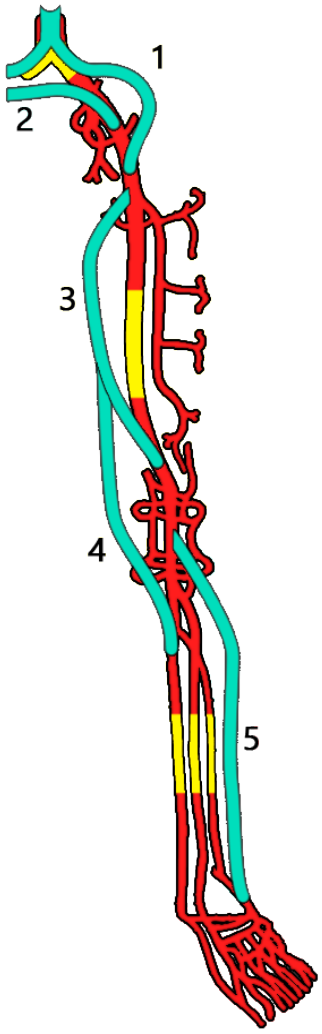
Nyní se krátce zaměříme na typické rekonstrukční terapie v kontextu ICHDK. Jako první uvedeme **aortofemorální bypass**, který se používá při stenóze pánevních tepen nebo aorty. Proximální anastomóza (typicky end-to-side) vystupuje z abdominální aorty, distální anastomóza je na femorální artérii. V případě oboustranného uzávěru pánevních tepen se přistupuje k použití bifurkačního aortobifemorálního bypassu. V případě úplného uzávěru se zakládá proximální anastomóza typu end-to-end [22]. V případě jednostranného uzávěru pánevního řečiště je možné implantovat i extraanatomické bypassy - **femoro-femorální cross-over bypass**, který spojuje zdravou femorální artérii s kontralaterální obliterovanou femorální artérií. **Axilo-bifemorální bypass** spojuje podpažní artérii (AA) s femorální artérií a následně s kontralaterální femorální artérií[17]. U takto dlouhých bypassů je však obzvláště důležité aktivně udržovat jejich průchodnost [22].

Polovina uzávěrů dolních končetin je způsobená obliterací povrchové femorální artérie (AFS) [18]. Při nedostatečnosti kolaterálního oběhu a endovaskulárních intervencí se přistupuje k implantaci **femoro-popliteálního bypassu**. Proximální anastomóza se našívá na společnou femorální artérii (AFC). Pro distální anastomózu je možnost volby nad a pod kolenním kloubem. Z tohoto důvodu rozlišujeme dále bypass **femoro-popliteální proximální** s distální anastomózou nad kolenem a bypass **femoro-popliteální distální** s distální anastomózou pod kolenem. Na AFS distálně navazuje zákolenní artérie (AP). Stenózy a uzávěry AP jsou typicky způsobené aterosklerózou. Krátké stenózy se dilatují endovaskulárně. Jako rekonstrukční zákroky slouží opět femoro-popliteální bypass nebo **popliteo-popliteální bypass**. AP je také druhým nejčastějším místem vzniku aneurysmat, hned po AAA. Při aneurysmatu popliteální artérie (AAP) existuje relativně značné riziko amputace končetiny, resp. končetin, neboť výskyt aneurysmat je typicky symetrický na obou dolních končetinách[18]. Léčba AAP je možná chirurgicky i endovaskulárně. Chirurgická léčba zahrnuje bypass vedoucí z AFS do místa pod výdutí na AP. Jako endovaskulární léčba se typicky používá stentgraft[18].

Z AP pod kolenem vystupují tři artérie: přední holenní artérie (ATA), zadní holenní artérie (ATP) a lýtková artérie (AF). Tyto tři artérie tvoří tzv. *bércové řečiště*. Stenózy se typicky vyskytují současně na všech třech artériích a často doprovází uzávěry AP nebo AFS. Chirurgickou léčbou je v tomto případě **femoro-pedální bypass** vedoucí z AFC do některé z tepen bércového řečiště nebo **bypass popliteo-pedální** vedoucí z AP do distální části ATA nebo některé z pedálních artérií. Průchodnost tepen bércového řečiště je důležitá také z pohledu efektivity bypassů implantovaných výše. Pro úspěšnou cévní rekonstrukci je totiž nezbytné zajistit dostatečný výtok z bypassu, což uzavřené bércové řečiště neumožňuje.

[17]Průtok v AA je velký a musí tak být rozveden do dobou dolních končetin.

[18]Pojem stentgraftu bude vysvětlen dále v samostatné podsekci.



Obrázek 2.11: Přehled umístění rekonstrukcí (zeleně) vzhledem k možným uzávěrům (žlutě). 1: aorto-bifemorální, 2: femoro-femorální cross-over, 3: femoro-popliteální proximální, 4: femoro-popliteální distální, 5: popliteo-pedální. Podklad obrázku převzat z `http://www.myfootshop.com/Content/Images/Anatomy/To_Add/Arteries_of_the_lower_extremities.jpg` [citováno 11. května 2016]

2.3.3 Materiál rekonstrukce

Jedním z faktorů, které ovlivňují výsledek cévní rekonstrukce, je volba použitého materiálu. Materiály dělíme na autologní a umělé cévní náhrady.

Autologní žíly

Jako autologní náhrada se většinou používá véna, zřídkakdy artérie. Pro tyto účely typicky slouží vena saphena magna (VSM), vena saphena parva (VSP) nebo vena cephalica (VC). Žilní štěp je považován za nejlepší cévní náhradu [18] a navíc má dobré mechanické vlastnosti, např. poddajnost nebo ohebnost. První nevýhodou autologní náhrady, kterou zmíníme, je fakt, že odpovídající žilní štěp nemusí být vždy dostupný. Druhou potenciální nevýhodou je nutnost nesmírně opatrné manipulace s žilním štěpem, aby nedošlo k poranění endotelu

a následným stenózám.

U autologních žilních náhrad dále rozlišujeme i směr uložení, tedy směr, kterým bude protékat krev vzhledem k původnímu směru toku krve vénou. Nutnost rozlišovat směr uložení plyne z anatomické odlišnosti artérií a vén. Vény v dolních končetinách mají chlopně, které umožňují tok krve pouze jedním směrem, což znemožňuje zpětný tok krve způsobený gravitací. Prvním a nejčastějším způsobem uložení je tzv. **reverzní**. Reverzní znamená, že véna je oproti svému přirozenému směru otočena o 180° frontálně. Výhodou tohoto uložení je zachování směru proudění krve. Nevýhodou je potenciální rozdíl průměrů[19] na anastomózách. To vychází z přirozeného faktu, že cévy se obecně distálním směrem zužují, takže při reverzním uložení, kde se proximální konec šije na distální a obráceně, může nastat problém.

Oproti tomu při **nonreverzním** uložení není problém s rozdílnými průměry na anastomózách, neboť se šije proximální konec na proximální a obdobně na distálním konci. Opačně zde však vzniká problém s prouděním krve v náhradě proti venózním chlopním. Chlopně se v tomto případě narušují tzv. valvulotomem za účelem zlepšení průtoku[20][18].

Pro úplnost ještě uveďme poslední variantu a to uložení **in situ**. Při tomto uložení se neexplantuje žilní štěp, ale anastomózy se šijí na vénu při jejím přirozením uložení.

2.3.4 Cévní protézy

Pokud nelze použít autologní žílu, používají se umělé cévní náhrady nazývané též cévní protézy nebo jednoslovně **grafty**[21]. Různé grafty se liší, kromě délky a kalibru, materiálem a vnitřní strukturou.

Nejběžnějšími materiály používanými pro cévní protézy jsou PET a ePTFE. Podle struktury dělíme grafty na **lité**, **tkané** a **pletené**. Protézy z PET, v literatuře se často setkáváme s alternativním pojmenováním **dacron**, jsou tkané nebo pletené. Tkané náhrady jsou pevné a relativně nepoddajné. Proto se používají v oblasti aorty. Pletené náhrady menšího kalibru se používají na periferní cévy[23]. Jejich stěna bývá napuštěna kolagenem, čímž se předejde propouštění krve. Pletené náhrady dále mohou být vyztuženy spirálou nebo kroužky, které vytváří tzv. **armování**. Armování, viditelné na obrázku 2.12, pomáhá předejít mechanické stenóze graftu. ePTFE protézy se používají jako lité. Lité protézy nepropouštějí krev a mají dobré hemodynamické vlastnosti, nicméně oproti tkaným nebo pleteným protézám jsou méně pružné. Existují i protézy kombinující oba materiály - vnitřní litou ePTFE a vnější pletenou PET vrstvu[22].

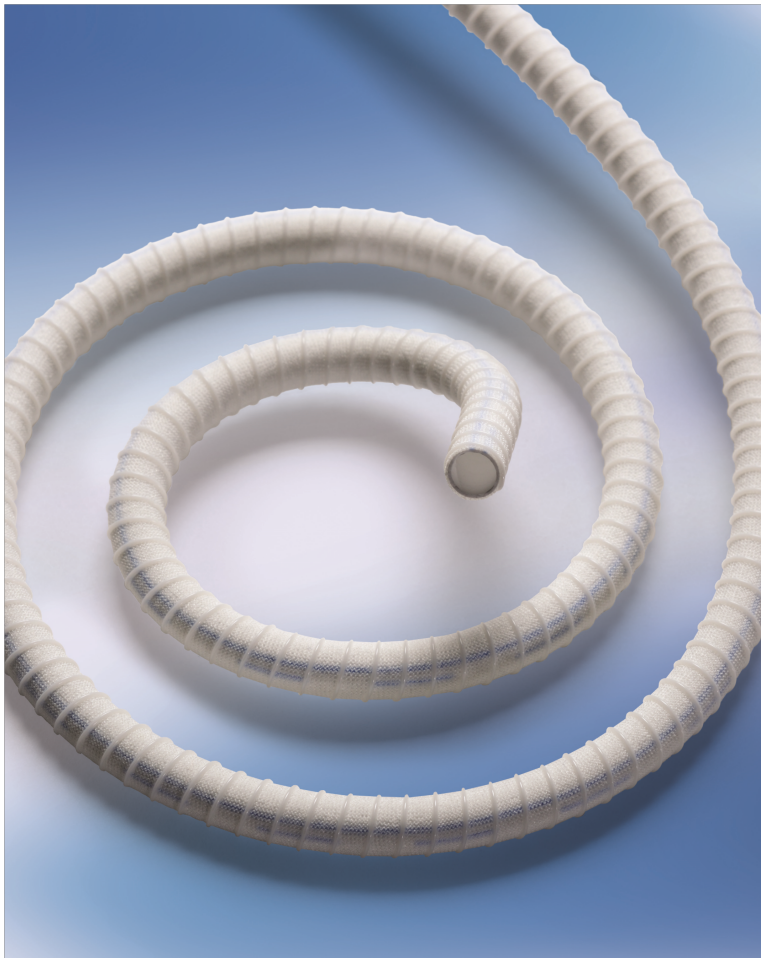
Stěny protéz je možné dále ošetřit a zlepšit tak jejich vlastnosti. Pro snížení rizika vzniku trombů je možné stěny impregnovat heparinem. Pro prevenci infekce se pak používá impregnace antibiotiky nebo solemi stříbra.

[19]Někdy se také průměr žilní náhrady nazývá „kalibr“.

[20]Valvulotom se používá i případě reverzního uložení. Pro názornost jej však zmiňujeme až při popisu nonreverzního uložení.

[21]Pojem graft může být poněkud přetížený. Může označovat i libovolnou cévní náhradu bez ohledu na materiál.

[22]Např. FUSION Vascular Graft: `http://www.maquet.com/int/products/fusion-vascular-graft/?ccid=191` [citováno 11. května 2016]

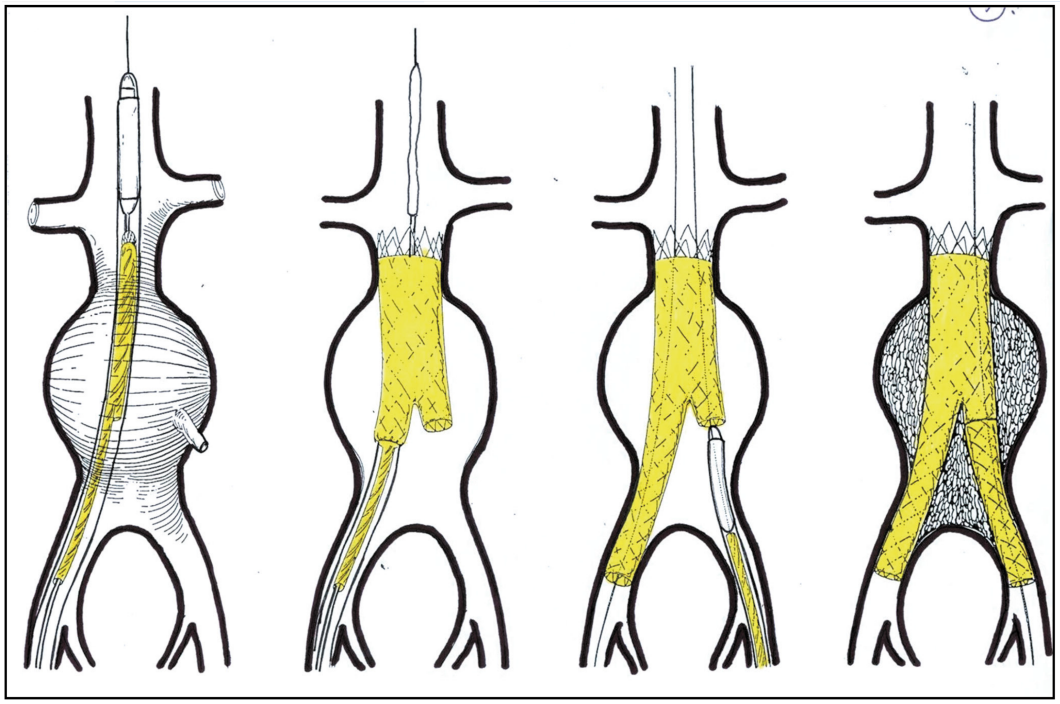


Obrázek 2.12: Pletená protéza Maquet fusion. Můžeme vidět zřetelné armování. Převzato z `http://www.maquet.com/int/products/fusion-vascular-graft` [citováno 11. května 2016]

2.3.5 Stentgraft

Na závěr ještě uvádíme jeden typ protézy, který jsme doposud nezmínili a to tzv. **stentgraft**. Již z názvu vyplývá, že se jedná o jakousi kombinaci stentu a graftu zároveň. Z vlastností stentu přebírá endoluminální umístění, z vlastností graftů pak nepropustnost krve. Stentgraft se využívá v situaci, kdy je potřeba krev vést stejnou artérií, nicméně bez kontaktu s její stěnou. Typickým příkladem použití je rekonstrukce aneurysmat. Stentgraftem se vede krevní tok skrz aneurysma, aniž by TK působil na stěnu výdutě, čímž se významně sníží riziko ruptury aneurysmatu. Stentgraft se v tomto případě upevní proximálně a distálně od výdutě v místě tzv. krčků aneurysmatu[18]. Oproti chirurgické terapii působí endovaskulární zavedení stentgraftu menší hemodynamické změny a tak je pro pacienty méně zatěžující.

Stentgrafty se nejčastěji používají při AAA. V tomto případě se typicky implantuje tzv. **bifurkační stentgraft**.



Obrázek 2.13: Postup implantace bifurkačního stentgraftu. Svinutý stentgraft se umístí pomocí katetru. Stentgraft se uvolní a rozvine se pomocí stentu, proximální anastomóza se zachytí v proximálním krčku aneurysmatu. Katetrem z kontralaterální AFC se zavede i druhá větev bifurkačního stentgraftu. Převzato z [18]

Proximální konec je umístěn v AA pod odstupem renálních artérií. Stentgraft se poté rozděluje do tvaru obráceného Y a kopíruje tvar bifurkace aorty. Distální zakončení jsou tedy dvě a to v oblasti pánevních artérií (proto se také setkáváme s označením aortobiiliakální stentgraft[18]).

2.4 Chronická renální insuficience

V následující sekci se nakrátko vzdálíme od vaskulární chirurgie, abychom se k ní vrátili po popisu domény druhé části zaměření naší práce.

Ledviny jsou párový orgán uložený v dutině břišní, jejichž primární funkcí je filtrování krve, při kterém se z ní odstraňují odpadní metabolické produkty a další škodlivé látky. Základní stavební jednotkou ledviny je **nefron**. Nefron obsahuje **glomerulus**, tvořený klubkem kapilár, ve kterých dochází k primární filtraci krve. Nefron se dále skládá z kanálku vedoucího do močového prostoru (tzv. **tubulus**). Ledvinami protéká každou minutu přibližně 20% minutového srdečního objemu[23]. Většina průtoku ledvinou je určena k filtraci a jen malá část ke krevnímu zásobení ledviny [8]. Filtrační činnost ledvin je pro život nezbytná, při kompletním selhání ledvin nastává smrt přibližně do pěti dnů.

Selhání ledvin (tzv. renální selhání) dělíme na akutní a chronické. Akutní selhání ledvin (ASL) je náhlý pokles funkce ledvin, který je často reverzibilní, tzn. léčbou je možné funkci ledvin plně obnovit. Bez dalšího upřesnění pro informaci uveďme, že příčiny ASL dělíme podle lokalizace na:

1. prerenální
2. renální
3. postrenální

V souvislosti s předchozí kapitolou zmíníme, že jedním z typů prerenální příčiny ASL je uzávěr renální artérie. Provedená cévní rekonstrukce (zejména v oblasti abdominální aorty) tak může mít zásadní vliv na funkci ledvin. Z pohledu naší práce se dále zaměříme na chronické selhání ledvin (CHSL). V literatuře se můžeme setkat se dvěma výklady tohoto termínu. V prvním případě jde o pojmenování všech chronických stavů, kdy ledviny již nedokáží filtrovat krev v potřebném množství, nicméně jsou částečně funkční. Pro tento stav však můžeme použít i specifičtější termín *chronická renální insuficience* (CHRI). V takovém případě pak CHSL označuje stav, kdy jsou ledviny již plně nefunkční. Průběh chronického onemocnění ledvin se podle National Kidney Foundation (NKF) dělí do několika stádií:

Stádium	**Popis**	**GF**
1	Normální funkce ledvin	>90
2	Mírná ztráta funkce ledvin	89-60
3a	Mírná až střední ztráta funkce ledvin	59-44
3b	Střední až závažná ztráta funkce ledvin	43-30
4	Závažná ztráta funkce ledvin	29-15
5	Terminální stádium, selhání ledvin	<14

Tabulka 2.1: Stádia chronické choroby ledvin podle NKF [24]

[23] Minutový srdeční objem (minutový srdeční výdej) je objem krve v litrech přečerpaný srdcem za jednu minutu[8].

[24] Url: `https://www.kidney.org/atoz/content/gfr` [citováno 11. května 2016]

Hodnota glomerulární filtrace (GF) slouží jako míra funkce ledvin. Lze ji určit z moči nebo krve na základě koncentrace látek, které se uvolňují právě při glomerulární filtraci.

Nejčastější příčinou CHSL, a to až ve 2/3 případů, je diabetes mellitus a arteriální hypertenze[24][25]. AH a DM totiž často způsobují **arteriolosklerózu**. Arteriolosklerózа je vedle aterosklerózy dalším typem arteriosklerózy, která postihuje arterioly. Nejvýraznější změnou při arteriolosklerózе je hypertrofie medie vedoucí ke zúžení lumina arterioly[26] a nejčastěji se vyskytuje právě v ledvinách. Mezi další příčiny CHSL patří:

Glomerulonefritídy
: Onemocnění glomerulu (glomerulopatie) způsobující zánětlivé změny glomerulů.

Intersticiální nefritída

Polycystické onemocnění ledvin

Vrozené abnormality
: Např. vezikoureterální reflux.

Dědičná onemocnění
: Např. Alportův syndrom.

2.4.1 Hemodialýza

V případě, že ledviny svou funkcí přestávají pokrývat potřebu filtrace krve, tedy přibližně od stádia 3b[27], je pro život pacienta nezbytné zajistit alternativní metodu filtrace krve. V dnešní době je nejpoužívanější metodou tzv. **hemodialýza**. Při hemodialýze (HD) je krev odvedena z těla pacienta do přístroje, dialyzátoru, kde se pomocí difuze a filtrace přes semipermeabilní dialyzační membránu vyčistí a vrací zpět do pacientova krevního oběhu[8]. Jedním z rizik HD je tvorba krevních sraženin v dialyzátoru v důsledku turbolentního proudění a kontaktu s cizím tělesem. Proto je typicky nasazena antikoagulační terapie, nejčastěji heparin[28].

Další komplikace vyplývají již ze samé podstaty HD. A tou je potřeba odvést netriviální množství krve a jeho opětovné navrácení. Musíme mít na paměti také fakt, že HD je prováděna 2×-3× týdně u chronicky nemocných pacientů po dobu až několika let. Řešením je nalezení vhodného tzv. **cévního přístupu**. Cévní přístupy rozdělujeme na dočasné a trvalé. **Dočasný cévní přístup** se používá pro HD při akutním selhání ledvin nebo po dobu, kdy není dostupný trvalý cévní přístup.

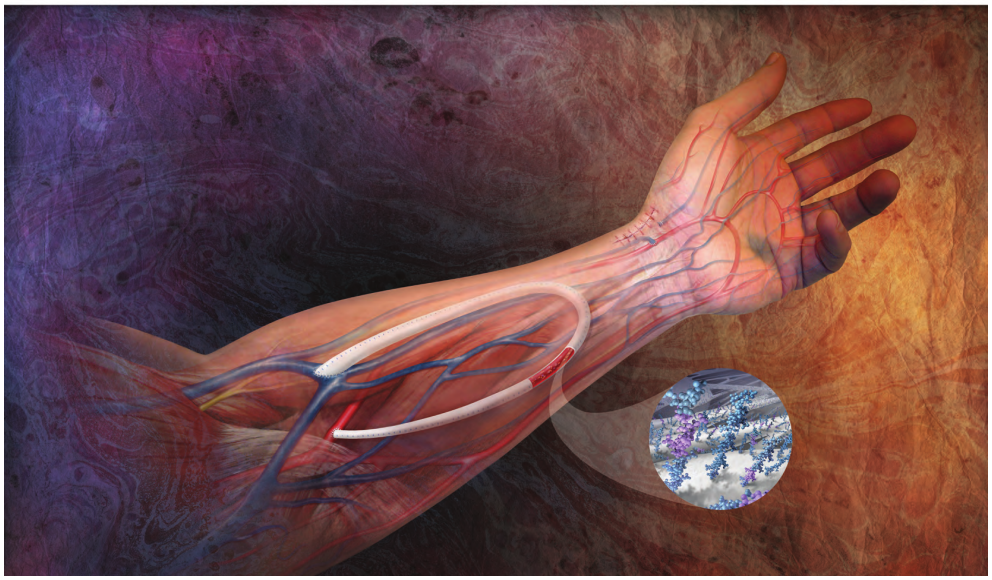
Jako **trvalý cévní přístup** se používá **arteriovenózní zkrat (AV zkrat)**, nazývaný též **arteriovenózní shunt** nebo **arteriovenózní píštěl**. Všechny tyto názvy budeme považovat za ekvivalentní. AV zkrat tvoří přímé propojení artérie s vénou. Vznikne tak místo s dobrým krevním zásobením - přívodem i odvodem. Žádoucí je také dostatečný průtok zkratem (alespoň 200 až 300 ml/min[8]). Po založení AV zkratu se vlivem zvýšeného průtoku mění stavba vény. Její průměr se

zvětšuje (dilatuje) a zároveň dochází ke ztluštění stěny zmnožením buněk hladké svaloviny (hypertrofie tunica media). Tomuto procesu se říká **arterializace**. Arterializace vény trvá zhruba 4-12 týdnů[29]. Pro toto období se také setkáváme s označením „zrání zkratu“. Zkrat je pro HD možné použít až po uplynutí tohoto období.

Nejčastější je radio-cefalický zkrat nedominantní ruky, tedy mezi arteria radialis a vena cephalica. Používá se typicky end-to-side anastomóza, ovšem je možné použít i end-to-end nebo side-to-side anastomózu. Pokud není možné umístit zkrat na zápěstí, postupuje se proximálně. Dalšími možnými zkraty jsou brachio-cefalický a brachio-basilický. V situacích, kdy nelze vytvořit přímý AV zkrat se využívají delší zkraty, skrz které je vedena krev z artérie do vény. Obdobně jako v případě cévních rekonstrukcí je možné použít biologický materiál (např. štěp VSM) nebo cévní protézu.

2.4.2 Protetický hemodialyzační přístup

Pokud není možné klasický zkrat vytvořit z důvodu nedostatečného cévního zásobení nebo když dříve založené zkraty již nevyhovují, přistupuje se k použití umělých cévních protéz, které umožňují vytvořit zkrat i mezi relativně vzdálenou artérií a vénou. Takovému zkratu se většinou říká **arteriovenózní graft**. Grafty se mimo radio-cefalických umisťují jako brachio-axilární, brachio-basilické, axilo-basilické nebo axilo-axilární[22]. V případě nutnosti je možné graft umístit i na dolní končetinu. Konkrétně jako femoro-safenózní nebo femoro-femorální[15]. Podle tvaru (resp. vedení) se grafty dělí na rovné (přímé) a oblouky. Jako materiál protézy se nejčastěji používá ePTFE, polyuretan nebo PET. Volba správného materiálu není dosud přesně stanovena, neboť zatím neexistuje studie, která by přinesla jednoznačné výsledky[30]. Oproti jednoduchým zkratům mají grafty výrazně kratší dobu zrání. Některé protézy lze použít již po 48 hodinách [30].



Obrázek 2.14: Vizualizace arteriovenózního graftu. Převzato z `http://www.goremedical.com/resources/dam/assets/AM0707EN1.PVG.BROCHURE.FNL.mr.pdf` [citováno 11. května 2016]

Hemodialyzační cévní přístupy, ať už jednoduché zkraty, nebo grafty, s sebou nesou jistá rizika komplikací[31]:

Stenóza
Stenóza patří mezi nejčastější komplikace zkratů. Až 80% stenóz způsobených turbolentním prouděním poškozujícím endotel se vyskytuje v oblasti venózní anastomózy. Stenóza může mít zásadní vliv na kvalitu cévního přístupu. Terapie je obvykle endovaskulární - PTA.

Trombóza
Společně se stenózou může dojít i k trombóze. Dalšími příčinami vzniku trombózy mohou být traumata nebo pokles arteriálního tlaku při použití přístupu pro HD. Řešení je možné farmakologicky, endovaskulárně i chirurgicky[32].

Srdeční selhání
Vzácně způsobené zvýšeným průtokem venózní anastomózou.

Steal syndrom
Pod tímto názvem je v literatuře označováno nedostatečné krevní zásobení končetiny distálně od zkratu. Projevuje se omezenou pohyblivostí, změnami barvy nebo bolestí. Může vést k ischemii až nekrózám. Důležité je proto vedle zkratu udržovat i dostatečný kolaterální oběh.

Aneurysma
Aneurysma může být způsobeno napichováním jednoho místa a to jak u žíly, tak u protetického graftu. Pokud zároveň nedojde k infekci, je možné provést resekci a implantaci nového graftu[15].

Infekce
Jako u každého chirurgického výkonu i zde existuje riziko infekce. Příčinou může být kontaminace při zakládání přístupu nebo pozdější napichování pro HD. Výskyt infekcí je u graftů častější než u jednoduchých zkratů. Léčba infekce je farmakologická podáním antibiotik. V krajním případě je nutné graft explantovat.

2.5 Shrnutí

V této kapitole jsme popsali doménu řešeného problému. Smyslem nebylo uvést všechny detaily, což by bylo mimo rozsah a zaměření této práce, ale dát prováděné výkony do souvislostí umožňující základní proniknutí do dané problematiky.

Doménu jsme rozdělili na cévní rekonstrukce a protetické hemodialyzační přístupy. Popsali jsme příčiny vzniku a rozvoj aterosklerózy jako nejčastější příčiny onemocnění artérií v takovém rozsahu, že je nutné přistoupit k chirurgické terapii. Dále jsme se omezili na cévní rekonstrukce artérií dolních končetin, kde jsme se také zabývali nejčastějšími lokalizacemi operací. Tuto část jsme uzavřeli rozdělením materiálů, které se používají pro cévní rekonstrukce. Zbývající část jsme věnovali protetickým hemodialyzačním přístupům neboli graftům. Na úvod

jsme popsali příčiny renálního selhání, kvůli jehož chronické formě se grafty zakládají. V poslední části jsme rozebrali typy graftů, možnosti jejich vedení a rizika, která jsou s jejich použitím spojená.

Ačkoli byla úvodní kapitola věnovaná téměř výhradně medicínským základům, podle našeho názoru je porozumění problematice nezbytné pro budoucí efektivní práci s daty. Kapitola může sloužit právě jako úvod do problematiky pro kohokoli, kdo by chtěl náš systém rozšířit nebo se věnovat analýze dat. Během přípravy této kapitoly jsme zjistili, že v současnosti dostupná literatura je buď zaměřena na laiky a tudíž příliš stručná, nebo naopak pro odborníky a až příliš detailní. V našem popisu domény se věnujeme pouze základním faktům, uvádíme je však v souvislostech, což umožňuje lepší pochopení problematiky.

Část II

Implementace

3. Implementace

V této kapitole se budeme věnovat softwarové části práce. Zabývat se budeme procesem vývoje našeho systému, architekturou, použitými technologiemi a některými implementačními detaily.

3.1 Požadavky

Byli jsme osloveni II. chirurgickou klinikou kardiovaskulární chirurgie Všeobecné fakultní nemocnice v Praze, abychom vytvořili aplikaci umožňující evidenci a analýzu dat cévních rekonstrukcí. Motivací z medicínského pohledu byla neexistence nástroje, který by pracoval s daty právě z této domény a umožnil jejich efektivní analýzu. Ta doposud nebyla možná, neboť záznamy nejsou uloženy v digitální podobě. Dalším motivem pro vývoj aplikace je tak i možnost digitalizace archivních záznamů.

3.1.1 Funkční požadavky

Požadavky se při vývoji softwaru dělí na funkční a obecné (technické). Tohoto dělení se budeme držet i my. Funkční požadavky specifikují základní vlastnosti softwaru z uživatelského pohledu. V literatuře se můžeme setkat s definicí funkčního požadavku jako *„popisu chování systému v závislosti na podmínkách“* [33, str. 7]. Funkční požadavky tak reflektují představu uživatele (zadavatele)[1] o tom, co bude software umožňovat. V případě mezioborové práce, jako je naše, je tak důležité uživatelské požadavky správně transformovat do funkčních požadavků, které je dále možné použít pro specifikaci vyvíjeného softwaru.

Základní funkcí aplikace je evidence záznamů a zobrazení dříve uložených záznamů. Evidenci záznamů je rozdělena do dvou kategorií - evidence arteriovenózních graftů (AVG) a evidence rekonstrukcí.

Evidence AVG

Evidence AVG se skládá z evidence pacientů, evidence graftů a evidence kontrol. K pacientům jsou evidovány AV grafty, k AV graftům jsou evidovány kontroly. V rámci kontroly AV graftu může být provedena chirurgická terapie, endovaskulární intervence nebo může být zjištěna infekce. Detaily požadavků shrnují následující tabulky:

Evidence pacientů	
Popis	V aplikaci bude možné evidovat pacienty.
	U pacienta evidujeme následující údaje: • číslo pacienta (povinný) • pohlaví (povinný) • rok narození (povinný)

[1] V dalším textu budeme pojmy uživatele a zadavatele považovat za ekvivalentní, pokud nebude explicitně řečeno jinak.

	• onemocnění (povinný) • diabetes mellitus • hyperlipidémie • arteriální hypertenze • začátek hemodialýzy (povinný) • datum provedené transplantace • datum úmrtí
Evidence graftů	
Popis	Pacientovi je možné evidovat AV grafty.
	U AV graftu evidujeme následující údaje: • číslo graftu (povinný) • datum založení (povinný) • datum definitivního uzávěru • lokalizace (povinný) • typ (povinný) • značka protézy (povinný)
Evidence kontrol graftů	
Popis	Pacientovi je možné evidovat kontrolu AV graftu.
	U kontroly AV graftu evidujeme následující údaje: • datum kontroly (povinný) • antikoagulační terapie • antiagregační terapie • výsledky ultrazvuku • indikace uzávěru graftu • poznámka
	Výsledky ultrazvuku zahrnují: • průměr a rychlost toku přívodní artérií • hodnoty průtoku graftem • reziduální průměr, rychlost toku a sílu intimy venózní anastomózy
Evidence chirurgické terapie	
Popis	V rámci kontroly AV graftu je možné evidovat provedenou chirurgickou terapii.
	U chirurgické terapie evidujeme následující údaje: • chirurgická revize žilní/arteriální anastomózy • plastika žilní/arteriální anastomózy • žilní/arteriální reanastomóza • trombectomie graftu
Evidence infekce	
Popis	V rámci kontroly AV graftu je možné evidovat zjištěnou infekci.
	Infekce evidujeme následující údaje: • původce infekce arteriální anastomózy/graftu/žilní anastomózy • parciální resekce graftu pro infekci • parciální resekce graftu pro PSA

	• explantace graftu
Evidence endovaskulární intervence	
Popis	V rámci kontroly AV graftu je možné evidovat provedenou endovaskulární intervenci.
	U endovaskulární intervence evidujeme následující údaje: • trombolýza • rekanalizace graftu • PTA/stent: arteriální anastomózy/žilní anastomózy/graftu/odvodní žíly/přívodní artérie/žilního centrálního řečiště

Tabulka 3.1: Funkční požadavky evidence AVG

Evidence rekonstrukcí

Evidence rekonstrukcí se skládá z evidence pacientů, evidence operací a evidence kontrol rekonstrukcí. V rámci operace evidujeme provedené PTA a použité stenty. V rámci kontroly rekonstrukce může být proveden chirurgický výkon, endovaskulární intervence nebo může být zjištěna infekce. Endovaskulární intervence zahrnuje evidenci provedených PTA a použitých stentů. Detaily požadavků shrnují následující tabulky:

Evidence pacientů	
Popis	V aplikaci bude možné evidovat pacienty.
	U pacienta evidujeme následující údaje: • číslo pacienta (povinný) • pohlaví (povinný) • rok narození (povinný) • ischemická choroba srdeční • diabetes • hyperlipidémie • arteriální hypertenze • kouření • chronická renální insuficience • ASA rezistence • datum provedené transplantace • datum úmrtí
Evidence operací	
Popis	V aplikaci bude možné evidovat rekonstrukční operace.
	U operace evidujeme následující údaje: • končetina, na které byla operace provedena (povinný) • číslo rekonstrukce (povinný) • datum operace (povinný) • umístění rekonstrukce (povinný) • pořadí operace (povinný) • použitý materiál (povinný)

	• provedené PTA/použité stenty
	U provedených PTA evidujeme: • artérii, na které byla provedena
	U použitých stentů evidujeme: • artérii, do které byl stent implantován • typ stentu
Evidence kontrol rekonstrukcí	
Popis	Pacientovi je možné evidovat kontrolu rekonstrukce.
	U kontroly rekonstrukce evidujeme následující údaje: • datum kontroly (povinný) • průchodné bércové tepny • průchodnost pedálního oblouku • antikoagulační terapie • antiagregační terapie • uzávěr a stenózu rekonstrukce • stenóza proximální a distální anastomózy
Evidence chirurgického výkonu	
Popis	V rámci kontroly rekonstrukce je možné evidovat chirurgický výkon.
	U chirurgického výkonu evidujeme následující údaje: • chirurgická revize proximální/distální anastomózy • chirurgická plastika proximální/distální anastomózy • trombectomie bypassu
Evidence endovaskulární intervence	
Popis	V rámci kontroly rekonstrukce je možné evidovat endovaskulární intervenci.
	U endovaskulární intervence evidujeme následující údaje: • PTA proximální anastomózy/rekonstrukce/distální anastomózy • trombolýza rekonstrukce/bércového řečiště • typ a lokalizace stentgraftu • provedené PTA/použité stenty
Evidence infekce	
Popis	V rámci kontroly rekonstrukce je možné evidovat zjištěnou infekci.
	U infekce evidujeme následující údaje: • původce infekce rekonstrukce • terapie infekce

Tabulka 3.2: Funkční požadavky evidence rekonstrukcí

Mimo tyto základní funkční požadavky umožňující zadávání dat byly na systém kladeny i další funkční uživatelské požadavky zohledňující různé typy uživatelů, zobrazení zadaných dat a administraci. Požadována je rovněž funkcionalita filtrování záznamů nebo výpočet statistických údajů:

Uživatelé	
Popis	Systém bude podporovat uživatele: • uživatel se bude moci přihlásit na základě jeho uživatelského jména a hesla • uživatelé budou mít různá oprávnění • uživatel s administrátorským oprávněním bude moci přidávat a editovat uživatele • uživatel bude moci změnit své heslo
Přehledy dat	
Popis	V systému budou zobrazeny přehledy dat: • přehled zadaných pacientů, kontrol AVG a kontrol rekonstrukcí pro daného uživatele • přehled pacientů • přehled kontrol AVG • přehled kontrol rekonstrukcí
	Přehledy bude možné filtrovat podle zadaných kritérií - podle názvu nebo rozsahu hodnot.
	V přehledech bude možné zjišťovat základní statistické údaje dat.
Správa číselníků	
Popis	Uživatel s administrátorským oprávněním bude moci editovat číselníky.
Export dat	
Popis	Data z aplikace bude možné exportovat do tabulek kompatibilních s procesorem Microsoft Excel 2007 a novějším.

Tabulka 3.3: Funkční požadavky zohledňující zobrazení dat, uživatele a administraci.

3.1.2 Obecné požadavky

Obecné požadavky[2] specifikují omezení a podmínky, za kterých softwarový systém splňuje funkční požadavky. Na systém byly kladeny následující obecné požadavky:

Prostředí	
Popis	Systém bude desktopová[3] aplikace běžící pod operačním systémem Microsoft Windows.

[2]V anglické literatuře se používá termín „nonfunctional requirement“. Překlad „nefunkční požadavek“ je v češtině poněkud zavádějící. Nicméně i s tímto českým názvem se můžeme v literatuře nebo praxi setkat.

[3]Pojem desktopové aplikace záměrně udáváme explicitně. V dnešní době je na specifikované platformě, tedy Microsoft Windows, možné používat i tzv. univerzální aplikace, které jsou od desktopových ve značné míře odlišné.

Popis	Podporován bude OS Microsoft Windows 7 a novější.
Dostupnost	
Popis	Evidence a data budou dostupné po síti z více počítačů.

Tabulka 3.4: Obecné požadavky kladené na systém.

3.2 Architektura

Na základě daných požadavků jsme mohli navrhnout systém, který bude poskytovat požadovanou funkcionalitu za podmínek určených obecnými požadavky.

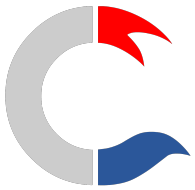
Základní struktury pro evidence AVG a rekonstrukcí znázorňuje diagram doménového modelu na obrázku 3.1.

Entity, které se vyskytují v obou evidencích (AVG i rekonstrukcí), jsme neduplikovali, ale používáme jednu entitu v obou kontextech.

Doménový model je nezávislý na technologii nebo platformě a proto se naší konkrétní implementaci začneme věnovat až v tento okamžik.

3.2.1 Systém Cerekon

Systém, který jsme implementovali, jsme se rozhodli pojmenovat **Cerekon**. Jedná se o akronym ze slov **cé**vní **rekon**strukce. Budeme-li v dalším textu hovořit o systému, bude se jednat právě o implementaci systému Cerekon.



Obrázek 3.2: Logotyp systému Cerekon.

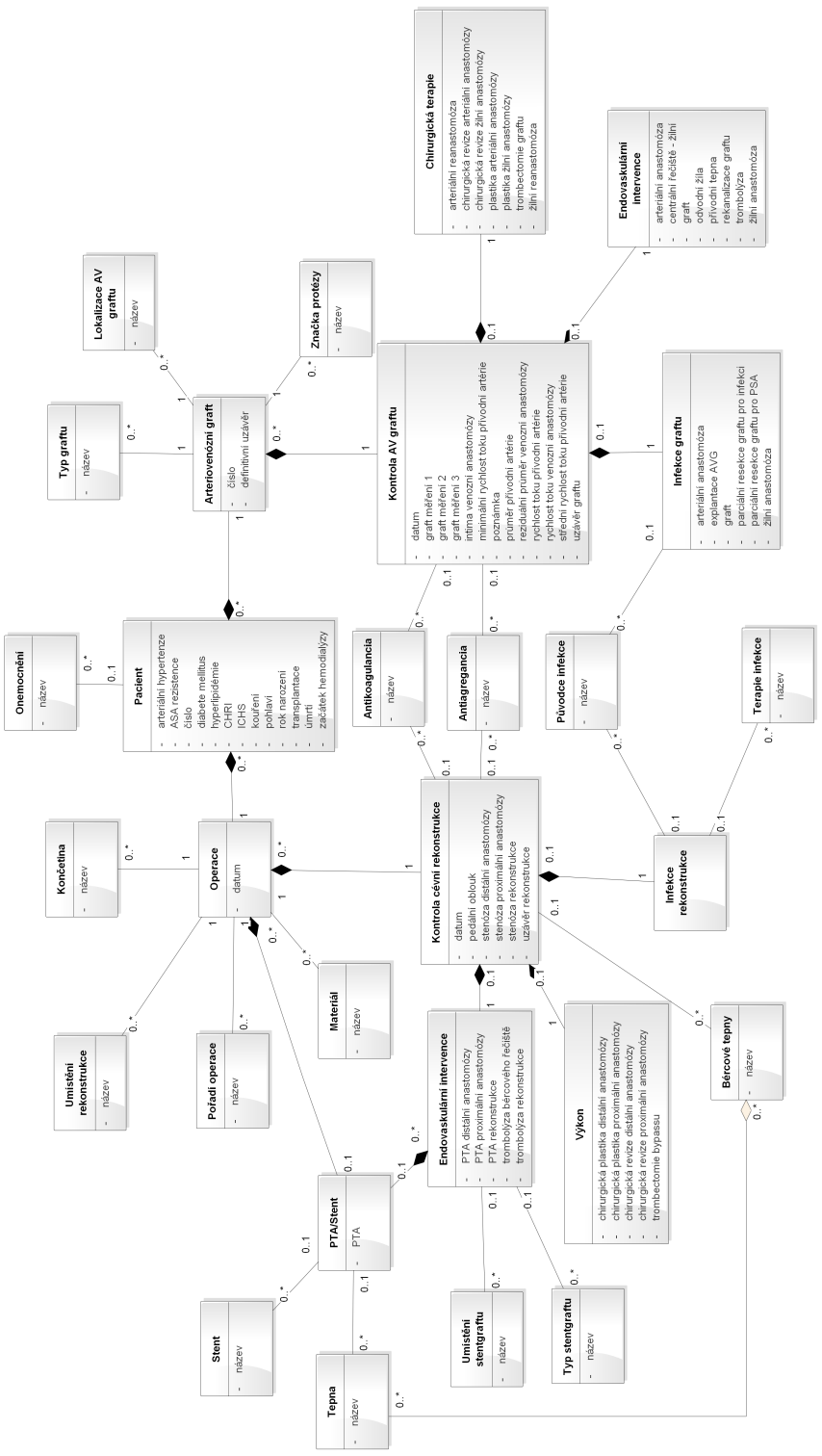
Cerekon je informační systém typu klient-server implementující vícevrstvou architekturu. To znamená, že jednotlivé funkční celky systému, tvořící tzv. vrstvy, jsou navzájem relativně nezávislé. V případě systému Cerekon se jedná konkrétně o třívrstvou architekturu[4], která se vyznačuje rozdělením na:

datovou vrstvu
: perzistentní úložiště zajišťující ukládání a načítání dat

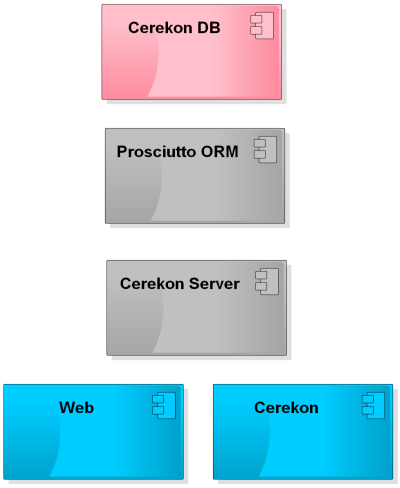
aplikační vrstvu
: vrstva obsahující hlavní aplikační logiku

prezentační vrstvu
: zprostředkovává výstup systému pro uživatelské rozhraní

[4]Některé součásti bychom mohli v závislosti na použité definici označit za samostatné vrstvy. Získali bychom tak čtyřvrstvý systém nebo jiný model vícevrstevné architektury. Pro jednoduchost však budeme považovat systém za třívrstvý a části, které bychom mohli označit jako samostatné vrstvy, zmíníme v popisu příslušné komponenty.



Obrázek 3.1: Doménový model evidence pacientů, graftů, rekonstrukcí a kontrol.



Obrázek 3.3: Rozdělení komponent systému Cerekon do vrstev. Červená - datová vrstva, šedá - aplikační vrstva, modrá - prezentační vrstva.

Rozdělení základních komponent systému do tří vrstev ilustruje obrázek 3.3. Rozdělení do vrstev odpovídá intuitivní představě databáze - server - klient.

V následujících kapitolách se budeme blíže zabývat jednotlivými částmi systému.

4. Databáze

Jako implementaci datové vrstvy systému jsme se rozhodli použít relační databázi (RDBMS). RDBMS poskytuje možnost flexibilního dotazování, silnou konzistenci a existuje řada nástrojů a postupů pro práci s RDBMS. Konkrétně jsme se rozhodli použít relační databázi Microsoft SQL Server[1][2] a to ve edici 2014 Express[3]. Tato edice je volně dostupná a z toho vyplývající limity (použití max. 1GB RAM, velikost databáze omezená na 10GB a využití nejvýše 4 CPU jader z jednoho socketu) pro naše použití prozatím nejsou zásadním způsobem omezující. SQL Server poskytuje možnost jednoduché správy, zálohování a obnovy databází a podporu jazyka Transact-SQL, procedurálního rozšíření jazyka SQL. Další výhodou je existence vhodných ovladačů pro připojení aplikací.

Entity z doménového modelu jsou tedy v logickém modelu (databázovém modelu) reprezentovány tabulkami fyzicky uloženými v SQL Serveru. Databázový model je normalizovaný podle třetí normální formy. Velkou výhodou normalizovaného modelu je omezení redundantního uložení dat. V podstatě se tak jedná o jednoduchou formu tzv. master data managementu (MDM). Použitím integritního omezení cizího klíče na referencovanou entitu (master data) místo redundantního lokálního uložení dat máme zaručenu konzistenci. V kontextu MDM se používá termín „jedna verze pravdy“, která je daná právě záznamem v referencované tabulce.

4.1 Konvence

Při návrhu a implementaci databázové vrstvy jsme se drželi některých zásad, které zvyšují efektivitu nebo zjednodušují použití databáze dalším vrstvám systému.

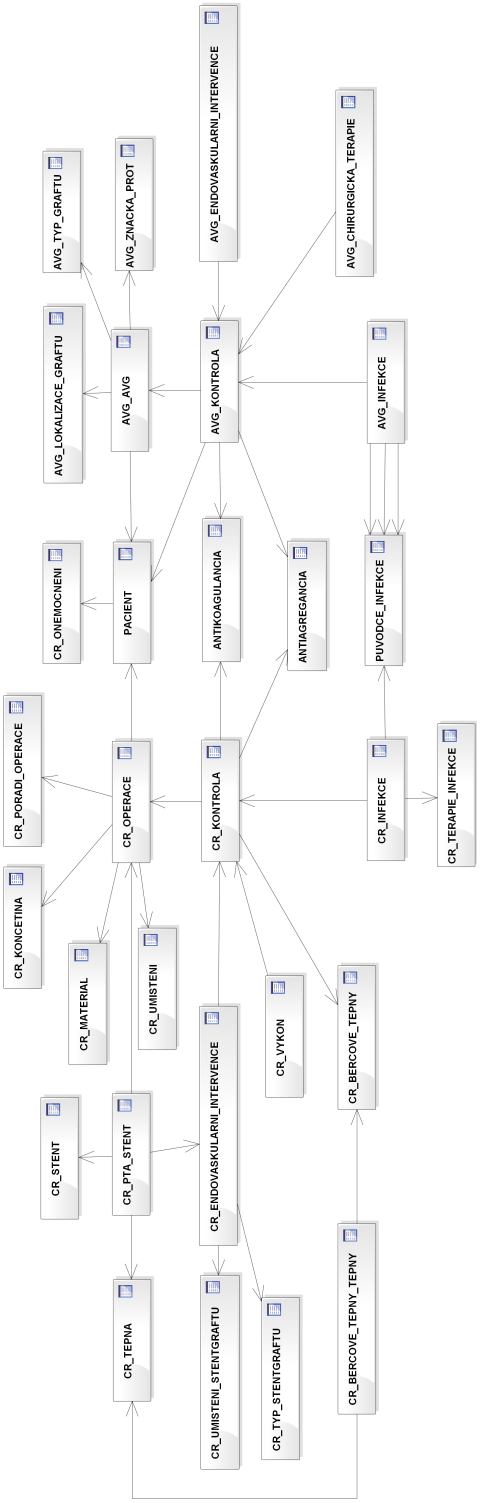
4.1.1 Jmenné konvence

Tabulky odpovídající entitám doménového modelu používají názvy odvozené z českých názvů entit doménového modelu (viz diagram na obrázku 4.1). V praxi se obecně používají anglické názvy entit, které bývají jednodušší a nehrozí možnost změny významu vypuštěním diakritiky a také práce s plurály je snazší. Takový problém v našem případě nenastal a pro české názvy v naší mezioborové práci jsme se rozhodli z důvodu snadnější komunikace s doménovými experty, tedy lékaři. Odpadá zde tak riziko možné chyby při překladu, neboť jsme při vývoji komunikovali v češtině. Důležitější je však složení názvů tabulek a sloupců. Název každé tabulky se skládá z prefixu označujícího logickou část modelu odděleného podtržítkem od názvu odvozeného z entity. Je tak možné jednoduše identifikovat, ke které části systému tabulka patří. Prefix se nepoužívá u tabulek, které se používají ve více logických částech systému.

[1] `https://www.microsoft.com/cs-cz/server-cloud/products/sql-server/` [citováno 11. května 2016]

[2] Microsoft SQL Server budeme v dalším textu označovat zkráceně jen jako SQL Server.

[3] `https://www.microsoft.com/cs-cz/server-cloud/products/sql-server-editions/sql-server-express.aspx` [citováno 11. května 2016]



Obrázek 4.1: Logický model databáze systému Cerekon.

Příklad:

Tabulka `PACIENT` se používá v evidenci AVG i rekonstrukcí, nemá tedy definovaný prefix.
Tabulka `AVG_TYP_GRAFTU` náleží pouze do evidence AVG.

Názvy sloupců se skládají z prefixu tabulky, je-li definovaný, a podtržítkem odděleným třípísmenným identifikátorem vycházejícím z názvu tabulky. Toto opět umožňuje jednoznačnou identifikaci sloupce v rámci celé databáze.

Příklad:

Sloupec `AVG_TYP_NAZEV` náleží do tabulky `AVG_TYP_GRAFTU` reprezentující název typu graftu. Všechny sloupce z této tabulky tak budou začínat prefixem `AVG_TYP_`.

Speciální jmennou konvenci jsme zavedli u číselníků. Za číselníky považujeme entity, jejichž hodnoty je možné z uživatelského pohledu identifikovat jejich názvem, resp. vyjadřují doménu kategoriální veličiny. Zavedená konvence říká, že každý číselník obsahuje sloupec, jehož název je složen z prefixu definovaného v předchozím odstavci a přidaného `NAZEV`.

4.1.2 Primární klíče

Zvláštní pozornost jsme také věnovali primárním klíčům, resp. sloupcům primárních klíčů. Nepoužíváme kompozitní klíče, tedy primární klíč každé tabulky je tvořen právě jedním sloupcem. Tento předpoklad zjednodušuje komunikaci s databází dalším vrstvám systému.

Dále jsme se rozhodli pro použití globálně unikátních identifikátorů, takzvaných GUID. GUID je 128-bitová hodnota, která se obvykle zapisuje ve formě hexadecimálního řetězce. Výhodou takového identifikátoru je jeho podpora v operačním systému Microsoft Windows a dalších aplikacích, například v SQL Serveru[4]. Výhodou takové podpory je možnost získání identifikátoru, který bude s vysokou pravděpodobností unikátní. Pokud bychom chtěli takovéto funkcionality dosáhnout s číselnými identifikátory, museli bychom využít implementaci nějakého globálního čítače, který by při každém přečtení vrátil unikátní hodnotu. Pro takový čítač by bylo možné použít tzv. sekvence[5], které SQL Server podporuje od verze 2012. Využití takového čítače je při implementaci složitější. Použití GUID identifikátoru navíc dává silnou možnost unikátní hodnoty generovat i mimo SQL Server[6]. Přirozenou nevýhodou je v porovnání s číselnými identifikátory velikost. Jak jsme již uvedli, GUID, v SQL Serveru implementovaný datovým typem `UNIQUEIDENTIFIER`, je v porovnání s typem `INT` 32× větší. Nicméně z hlediska předvídaných objemů dat v našem systému nepředpokládáme, že by velikost GUID hrála zásadní roli. Další komplikací spojenou s GUID je efektivita dotazování. V praxi a literatuře se často setkáváme se sloupcem primárního klíče definovaným obdobně jako:

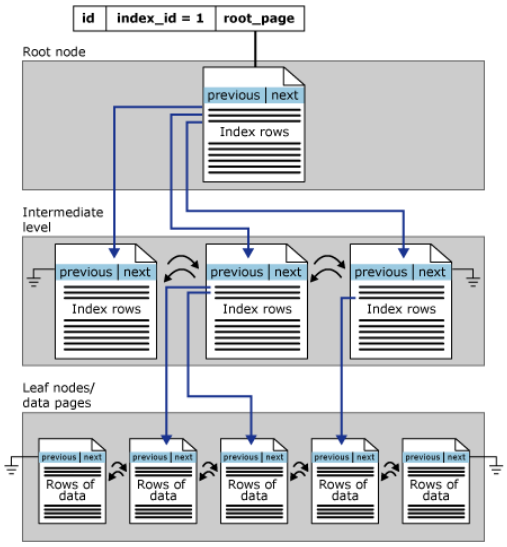
[4]SQL Server sám používá podporu z OS Microsoft Windows.

[5]`https://msdn.microsoft.com/en-us/library/ff878091.aspx` [citováno 11. května 2016]

[6]Nejčastěji pro tuto vlastnost se GUID identifikátory v praxi používají.

```
1 NAZEV_SLOUPCE INT IDENTITY(1,1),
2 CONSTRAINT NAZEV_KLICE PRIMARY KEY (NAZEV_SLOUPCE)
```

Taková definice představuje v SQL Serveru efektivní řešení. SQL Server při definici primárního klíče automaticky, není-li explicitně uvedeno jinak, vytváří klastrovaný index (CLUSTERED INDEX)[7] nad sloupcem primárního klíče. Efektivita tohoto řešení vyplývá z podoby klastrovaného indexu v SQL Serveru.



Obrázek 4.2: Schéma organizace dat v klastrovaném indexu SQL Serveru.

V listové úrovni klastrovaného indexu jsou uloženy datové stránky tabulky seřazené podle hodnot sloupce primárního klíče. Pokud tedy přidáváme záznam do tabulky s primárním klíčem definovaným výše, zapisujeme vždy na konec listové úrovně indexu. Takový zápis je typicky efektivnější než zápis na jiná místa indexu. Více o indexech v SQL Serveru je možné nalézt v [34].

Pokud bychom takto použili místo typu `INT` typ `UNIQUEIDENTIFIER`, zapisovali bychom při vkládání záznamu do náhodného místa v indexu. V SQL Serveru je implementována funkce `NEWSEQUENTIALID`[8], kterou je možné použít pouze jako implicitní hodnotu primárního klíče a která generuje uspořádané GUID hodnoty. Pokud do tabulky zapíšeme záznam, který má již definovanou hodnotu primárního klíče (tedy pokud vygenerujeme GUID hodnotu mimo tabulku a dokonce mimo databázi), opět dojde k potenciálně neefektivnímu zápisu na náhodné místo indexu.

Pro řešení výše popsaného problému využíváme dvou sloupců. Sloupec primárního klíče je typu `UNIQUEIDENTIFIER` a integritní omezení primárního klíče je definované jako `PRIMARY KEY NONCLUSTERED`. Dále pak v každé tabulce definujeme sloupec s názvem (kromě všech prefixů) `ORDINAL` typu `INT` a funkcí `IDENTITY(1,1)` a nad tímto sloupcem definujeme klastrovaný index tabulky. Pro další zlepšení efektivity dotazování, eliminací tzv. clustered lookup operací, je pak možné vytvořit ještě sekundární (NONCLUSTERED) index nad primárním

[7] `https://msdn.microsoft.com/en-us/library/ms190457.aspx` [citováno 11. května 2016]

[8] `https://msdn.microsoft.com/en-us/library/ms189786.aspx` [citováno 11. května 2016]

klíčem, který bude pokrývat i ostatní sloupce tabulky (tzv. covering index) pomocí klíčového slova `INCLUDE`[9]. Nevýhodou je pak režie spojená s údržbou dalšího indexu.

Poslední nevýhodou spojenou s použitím typu `UNIQUEIDENTIFIER` pro sloupec primárního klíče namísto funkce `IDENTITY` je nemožnost použít funkci `SCOPE_IDENTITY`[10]. Některé frameworky nebo knihovny nefungují s `UNIQUEIDENTIFIER` klíči právě z tohoto důvodu. Možnou alternativou je v SQL Serveru použití klauzule `OUTPUT`[11]:

```
1 INSERT INTO TABULKA (PRIMARNI_KLIC)
2 OUTPUT inserted.PRIMARNI_KLIC
3 VALUES ('12345678-1234-1234-1234-1234567890AB')
```

Nevýhodou takového řešení je např. nemožnost použití triggerů.

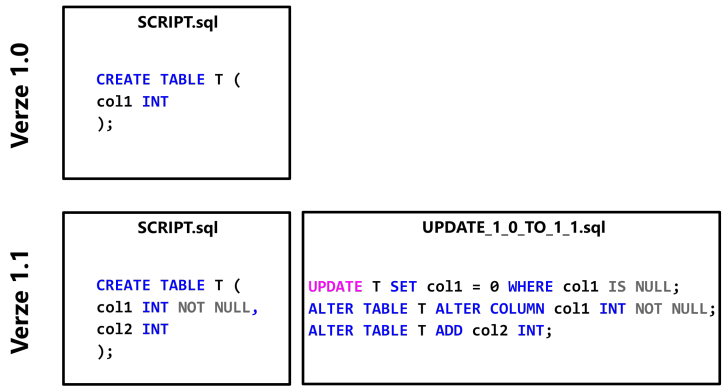
4.2 Správa verzí databáze

Verzování databází (perzistentního úložiště) je na rozdíl od jiných (volatilních) částí systému složitější. Zatímco u aplikačních částí systému stačí verzovat zdrojové kódy, které se kompilují, u databáze ve většině případů nestačí pouze verzovat příslušné DDL skripty. Pro databázi systému Cerekon jsme se rozhodli použít systém transformačních skriptů. V aktuální revizi jsou vždy uloženy pouze aktuální verze všech potřebných DDL a DML skriptů. To znamená, že pro vytvoření nové databáze systému stačí pouze spustit skripty podle manuálu. Ke každé změně v DDL nebo DML skriptu je, pokud je to nezbytné, vytvořen i transformační skript (viz obrázek 4.3), který po spuštění nad databází v předchozí verzi provede transformaci takovou, aby struktury odpovídaly strukturám v nové verzi. Poměrně zřejmou výhodou tohoto systému je, společně s použitím některého ze standardních systémů pro správu verzí (např. GIT, SVN, atp.), možnost ze zálohy databáze jednoduše vytvořit databázi v libovolné novější verzi.

[9] `https://msdn.microsoft.com/en-us/library/ms190806.aspx` [citováno 11. května 2016]

[10] `https://msdn.microsoft.com/en-us/library/ms190315.aspx` [citováno 11. května 2016]

[11] `https://msdn.microsoft.com/en-us/library/ms177564.aspx` [citováno 11. května 2016]



Obrázek 4.3: Schéma verzování databáze. Příklad zobrazuje dvě revize - 1.0 a 1.1. Ve verzi 1.1 navíc přibude transformační skript zakládající nově vytvořený sloupec.

5. Server

Serverová část systému Cerekon, kterou budeme nazývat jako „aplikační server“ nebo zkráceně jen „Server“, se skládá z několika částí, které postupně představíme:

- Prosciutto ORM
- Cerekon Common
- Cerekon Server Interfaces
- Cerekon Service Library

Dle požadavků má systém pracovat na operačním systému Microsoft Windows ve verzi 7 a výše, což nám garantuje možnost použít platformu Microsoft .NET[1], pro kterou jsme se rozhodli. Pro vývoj softwaru na platformě .NET je k dispozici více programovacích jazyků. Systém Cerekon jsme se rozhodli vyvíjet v jazyce C#, konkrétně ve aktuálně nejnovější verzi C# 6.0. Tato, pořád ještě relativně nová, verze jazyka přináší zajímavá vylepšení. Pouze ve stručnosti popíšeme nové konstrukce jazyka, které jsme se rozhodli ve větší míře využít.

5.1 C# 6.0

Předchozí verze jazyka, tedy C# 5.0, přinesla netriviální novou funkčnost a pouze malé změny v jazyce. Typickým příkladem je nová koncepce asynchronního programování pomocí klíčových slov `async` a `await`[2]. Oproti tomu verze 6.0 nepřináší tak zásadní změny funkčnosti, ale spíše více změn v jazyce, díky kterým je možné psát jednodušší kód.

Using static

První novou vlastností jazyka je spojení klíčových slov `using static` následované plným identifikátorem třídy. Veřejné statické metody takto referencované třídy je pak možné použít bez nutnosti psát název třídy. Pro příklad použijme třídu `System.Math`[3].

Příklad Pro výpočet délky přepony podle Pythagorovy věty bychom psali:

```
var c = Math.Sqrt(Math.Pow(a, 2) + Math.Pow(b, 2));
```

C# 6.0 umožňuje novou konstrukci:

```
using static System.Math;
...
var c =  Sqrt(Pow(a, 2) + Pow(b, 2));
```

[1] `https://www.microsoft.com/net` [citováno 11. května 2016]

[2] `https://msdn.microsoft.com/cs-cz/library/hh191443.aspx` [citováno 11. května 2016]

[3] `https://msdn.microsoft.com/cs-cz/library/system.math(v=vs.110).aspx` [citováno 11. května 2016]

Kód, kde se často pracuje s matematickými výpočty pak bude jednodušší a přehlednější.

Null propagation

V C# 6.0 je možnost testování reference na `null` hodnotu pomocí klasické podmínky `if` nahradit použitím nového operátoru `?.`, který se také nazývá jako „null propagation operator“. V programátorské komunitě je možné se setkat s označením „Elvis operator“. Kód zapsaný za operátorem se vykoná tehdy a jen tehdy, pokud je před operátorem reference různá od `null`.

Příklad V předchozích verzích jazyka bychom použili obdobnou kontrukci:

```
if (instance != null)
{
        instance.DoWork();
}
```

Ekvivalentní zápis s použitím null propagation:

```
instance?.DoWork();
```

String interpolation

Často používanou operací je skládání a formátování řetězců. Některé řetězce jsou konstanty, některé jsou proměnné. Jedním z často používaných řešení je pomocí metody `String.Format`[4]. Nevýhodou tohoto řešení je špatná čitelnost kódu zhoršující se úměrně s počtem skládaných řetězců a také nutnost dodržovat pořadí řetězců. Nová vlastnost jazyka, tzv. „string interpolation“, umožňuje skládání zahrnout přímo do řetězce v kódu. Stačí před začátek daného řetězce přidat znak `$` a vložené části řetězce zapsat do složených závorek. Lepší vysvětlení poskytne následující příklad:

Příklad Původní zápis pomocí `String.Format`:

```
var s = String.Format("Test person: {0} - {1}."
,P.FirstName,P.LastName);
```

S využitím string interpolation:

```
var s = $"Test person: {P.FirstName} - {P.LastName}.";
```

Nameof

Poslední novou vlastností jazyka, o které se zde zmíníme, je funkce `nameof`. Funkce `nameof` má jeden argument, kterým může být typ, proměnná nebo prvek třídy. Návratovou hodnotou je pak název argumentu. Výhodou použití této funkce je kontrola použitých jmen kompilátorem. Pokud tedy např. přejmenujeme prvek třídy a nepřejmenujeme jeho použití ve funkci `nameof`, kompilace skončí chybou.

[4] `https://msdn.microsoft.com/cs-cz/library/system.string.format(v=vs.110).aspx` [citováno 11. května 2016]

Taková chyba je žádoucí, neboť se již dále nepropaguje, což by se při použití jednoduchého řetězce obsahujícího název prvku stalo. Typickými příklady použití je při implementaci rozhraní `INotifyPropertyChanged`[5].

Příklad Následující ukázka kódu uloží do proměnné `nameProperty` řetězec `"FirstName"`.

```
var nameProperty = nameof(Person.FirstName);
```

5.2 Prosciutto ORM

V systémech, kde je datová vrstva implementována v relační databázi a aplikační vrstva v objektovém imperativním jazyce, se často využívá mechanismus objektově-relačního mapování (ORM). Úlohou ORM je mapování entit relační databáze na třídy objektového jazyka a naopak. ORM obvykle také poskytuje nástroje pro dotazování a zapisování dat. Na aplikační úrovni se pak pracuje pouze s objekty a aplikační programátor je odstíněn od práce s databází.

Pro jazyk C#, potažmo platformu .NET, existuje řada ORM frameworků. Mezi nejznámější patří NHibernate[6] a hlavně Entity Framework[7]. Entity Framework je v současné době patrně nejpoužívanější ORM framework pro platformu .NET. Pro systém Cerekon jsme se však po úvaze rozhodli implementovat vlastní ORM framework. Mezi důvody pro vývoj vlastního ORM systému jsou:

- Lepší kontrola nad databázovými dotazy, které ORM generuje.
- Možnost jednoduše definovat vlastní typy mapování.
- Možnost integrace pokročilých vlastností SQL Serveru.
- Možnost integrace databáze s externími (např. analytickými) nástroji již na úrovni ORM.
- Absence dalších pomocných souborů použitých pro mapování, které je nutné spravovat.

Všech uvedených vlastností bychom mohli dosáhnout i přidáním další vrstvy nad Entity Framework. Tím bychom však stejně ztratili většinu výhod, které jeho použití přináší. Alternativou by byla i úprava Entity Frameworku, neboť je jeho zdrojový kód otevřený. Udržování kompatibility upravených částí s ostatními částmi frameworku by bylo velmi náročné.

Náš ORM framework, který jsme pojmenovali „*Prosciutto*“, sestává ze dvou částí:

Prosciutto.Public
: Část obsahující struktury pro použití ORM ve vyšších vrstvách systému.

Prosciutto.ORM
: Část obsahující samotnou implementaci ORM.

[5] `https://msdn.microsoft.com/cs-cz/library/system.componentmodel.inotifypropertychanged(v=vs.110).aspx` [citováno 11. května 2016]

[6] `http://nhibernate.info/` [citováno 11. května 2016]

[7] `https://msdn.microsoft.com/en-us/data/ef.aspx` [citováno 11. května 2016]

Prosciutto.Public

Tato komponenta obsahuje potřebná rozhraní a struktury pro vytvoření mapování modelu, dotazování a ukládání objektů, logování a zpřístupnění dalších funkcí. Popis všech dostupných funkcí je mimo zaměření tohoto textu, zmíníme alespoň základní principy použití frameworku.

Mapování Pro definici mapování jsme zvolili tzv. atributy[8]. Atribut, v některých jazycích označovaný jako „anotace“, se zapisuje nad prvek kódu (třídu, metodu, pole) a obvykle nese rozšiřující informace (metadata) o daném prvku. Právě takovými metadaty jsou i informace o tom, že třída odpovídá tabulce, pole sloupci, atp. Následující příklad definuje třídu `Test`, která je mapována na tabulku `TEST_SCHEMA.TEST_ENTITY_TABLE`, která má primární klíč nad sloupcem `TEST_ID` a hodnoty tohoto sloupce nejsou generovány automaticky.

```
[Entity("TEST_ENTITY_TABLE","TEST","TEST_SCHEMA")]
class Test : IEntity<Guid>
{
    [Column("ID")]
    [PrimaryKey(false)]
    public Guid Id { get; set; }
}
```

Manipulace s daty Pro manipulaci s daty (jejich získávání a ukládání) využíváme návrhový vzor *repozitář* (repository pattern). Tento návrhový vzor říká, že veškerá manipulace s daty jednoho typu by měla být řešena v rámci systému na jednom místě. A právě tím místem je repozitář. Repozitář poskytuje rozhraní pro získání jednoho nebo více záznamů, ukládání a mazání. Repozitáři podporujícímu tyto operace se také někdy říká CRUD repository. Název vychází z operací create, read, update a delete. Zařazení repozitáře do systému zachycuje obrázek 5.1.

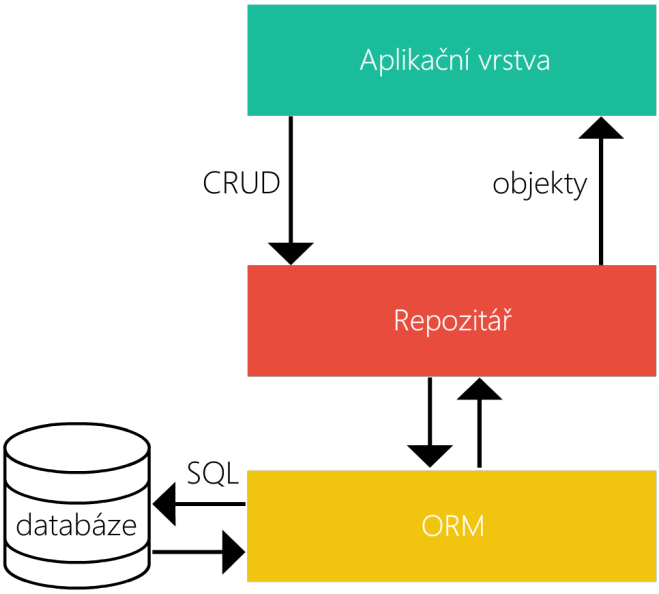
Zkratka CRUD může být trochu zavádějící. Explicitní vyjádření operací Create a Update by značilo, že typické rozhraní repozitáře bude podporovat, mimo jiné, právě tyto dvě metody. Dle našeho názoru je naopak žádoucí, aby operace Create a Update byly skryty za jedinou operací Save a rozhodnutí o uložení nového záznamu nebo aktualizaci existujícího bylo v režii repozitáře.

Dotazování Mezi základní operace s daty patří dotazování podle různých kritérií[9]. V Prosciutto ORM používáme objekty dotazů. Dotaz je jednoduchá struktura definující omezující podmínky, které musí datové záznamy splňovat, aby byly zahrnuty do výsledků. Konkrétně se jedná o:

1. počet záznamů, které se mají ignorovat
2. počet záznamů, které se mají zahrnout do výsledků
3. směr a atributy, podle kterých má být výsledek seřazen

[8] `https://msdn.microsoft.com/en-us/library/system.attribute(v=vs.110).aspx` [citováno 11. května 2016]

[9] Některé ORM frameworky tuto funkcionalitu nazývají přímo „Criteria“.



Obrázek 5.1: Návrhový vzor Repozitář.

4. odkaz na kořen stromu reprezentujícího výraz omezení dat

Posledně jmenovaný abstraktní strom výrazu nazýváme jako „selekci“. Selekce je stromová struktura skládající se operandů a operátorů. Operátory jsou rozděleny na unární, binární, ternární a n-ární. Listy jsou vždy konstanta nebo prvek reprezentující atribut. Naše implementace zahrnuje většinu standardně používaných operátorů. V případě potřeby by nebylo složité implementovat další rozšíření mechanizmu selekcí. Jednou z možností rozšíření je implementace databázové funkce provádějící netriviální operace s daty přímo v databázi. Na aplikační úrovni stačí posléze implementovat odpovídající operátor podle příslušného rozhraní a jeho správný převod na volání dříve vytvořené databázové funkce. Bylo by takto možné implementovat např. podobnostní vyhledávání v databázi, které by se na aplikační úrovni používalo jako standardní operátor.

Pro jednoduché vytváření selekcí jsme připravili třídu obsahující statické tovární metody vytvářející jednotlivé operátory nebo operandy. Zvláště pak ve spojení s výše popsaným `using static` je použití jednoduché, výsledný kód je relativně velmi dobře čitelný a s použitím `nameof` snadno udržovatelný. Uveďme jeden příklad: vytvoření selekce pacientů, kteří se narodili po roce 1950 nebo nekouří.

```
using static Prosciutto.Public.Querying.
        Factory.StaticSelectionFactory;
...
Or(
        Geq(nameof(Pacient.RokNarozeni), 1950),
        Eq(nameof(Pacient.Koureni), false)
);
```

Logování Logování, tedy zaznamenávání událostí, není nedílnou součástí ORM frameworků. My jsme se jej rozhodli implementovat, protože jde o užitečnou funkci, jejíž implementace není náročná a navíc je možné použít již existující

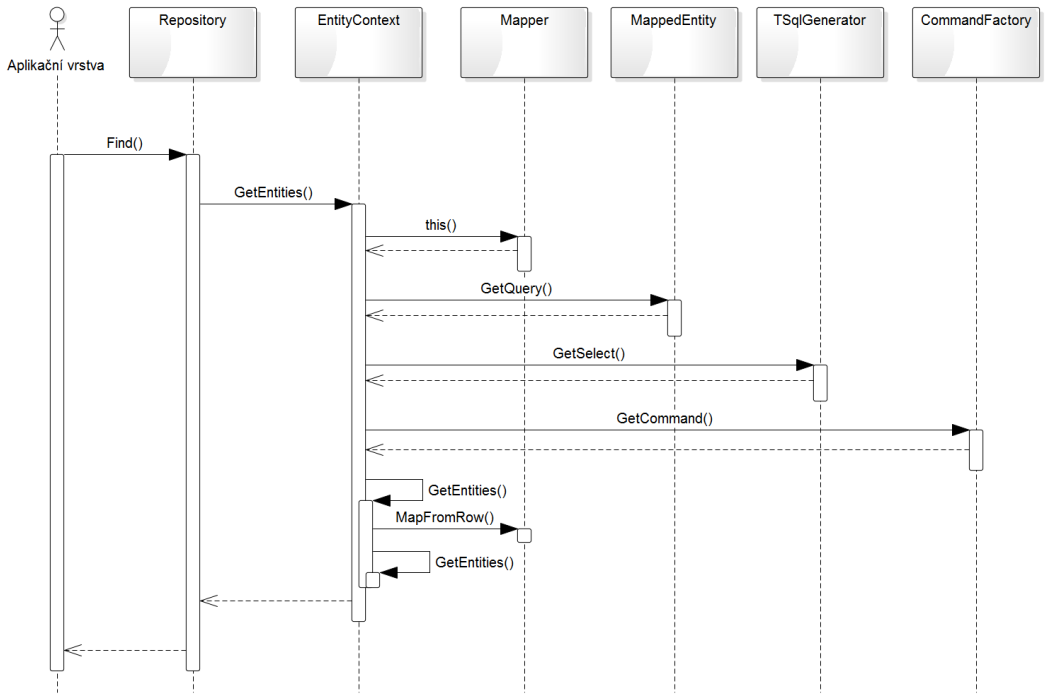
funkcionalitu, která je pro základní fungování ORM nezbytná. Náš systém logování zaznamenává následující údaje:

- identifikátor záznamu
- identifikátor uživatele
- provedená akce
- časové razítko
- kolekce provedených změn

Záznamy je možné použít na aplikační úrovni - například uživateli se zobrazí jen ty záznamy, které vytvořil nebo upravil. Dalším možným využitím je rekonstrukce všech změn stavu záznamu v čase, což může být velmi přínosné pro analýzu dat.

Prosciutto.ORM

Samotná implementace ORM frameworku Prosciutto se nachází právě v této komponentě, která je interně dále rozdělena na rozhraní a jejich implementaci. Hlavní vnitřní struktury ilustrujeme na příkladu. Zjednodušený sekvenční diagram na obrázku 5.2 znázorňuje postup vyřízení požadavku na čtení dat.



Obrázek 5.2: Sekvenční diagram načítání dat v ORM.

Repozitář předá požadavek `EntityContextu`. `EntityContext` od `Mapperu` získá metadata popisující mapování požadovaného typu. Takové údaje poskytuje `MappedEntity`. `MappedEntity`, jelikož má všechny potřebné informace o mapování, vytvoří abstraktní strom reprezentující dotaz. Tento abstraktní strom zahrnuje spojení se všemi (i rekurzivně) asociovanými entitami. `TSqlGenerator`

vygeneruje z abstraktního stromu ekvivalentní dotaz v jazyce Transact SQL. TSQL dotaz se odešle do databáze v příkazu, který je získán z tovární metody. `EntityContext` iteruje přes získané výsledky (prvky databázové relace) a `Mapper` provádí jejich mapování na požadované objekty. V případě, že požadovaná entita definuje asociaci typu 1:N, je sestaven příslušný dotaz na asociované záznamy a postupuje se rekurzivně. Po zpracování všech záznamů se kolekce výsledků vrací repozitáři, resp. aplikační vrstvě.

Jak jsme v předchozím odstavci naznačili, definované asociace se všechny načítají implicitně. Takový přístup se obecně nazývá „eager loading“. Opačným přístupem je tzv. „lazy loading“, kdy se asociované objekty načítají na základě explicitního požadavku. Rozšíření ORM o lazy loading je jednou z našich priorit v dalším vývoji. Situace více vyhovující eager nebo lazy loadingu podle našich měření také tvořily největší rozdíly ve výkonnosti Prosciutto ORM frameworku a Entity Frameworku. V jiných situacích byly oba frameworky srovnatelné.

Vnořené objekty V rámci Prosciutto ORM jsme implementovali i možnost definovat atributy komplexního typu. Takové atributy teoreticky porušují první normální formu. V praxi není jejich správné použití komplikací, neboť se z pohledu databáze jedná o řetězec (serializovaný objekt) a komplexní struktura se používá až v objektovém prostředí. Díky přesunutí deserializace do ORM odpadá nutnost explicitní řešení deserializace pro všechny takové atributy.

5.3 Aplikační server

Aplikační server (AS), zkráceně pouze *server*, obsahuje aplikační logiku a poskytuje sdílený přístup k datům. V implementaci systému Cerekon předpokládáme vždy právě jednu běžící instanci serveru, ke které se připojují instance klientů. V této části se budeme věnovat vnitřní struktuře aplikačního serveru. Na úvod zmíníme některé z principů a technologií, které jsme při vývoji AS využili.

5.3.1 Použité technologie

Windows Communication Foundation

Pro implementaci komunikace AS s klienty jsme použili framework Windows Communication Foundation (WCF)[10]. WCF poskytuje podporu pro vytváření tzv. služeb (*services*), které je možné vzdáleně nebo lokálně využívat. Jedná se o implementaci aplikací, které jsou tzv. orientovány na služby (*service-oriented*). Podrobný popis architektury WCF je mimo zaměření naší práce, proto zmíníme pouze základní koncepty a vlastnosti WCF.

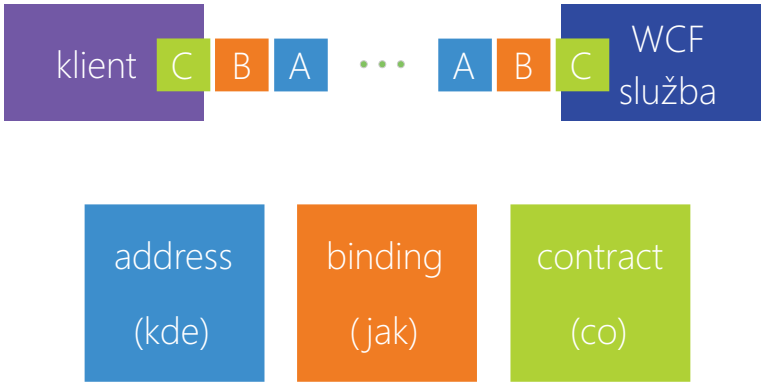
Pro komunikaci klienta se službou v prostředí WCF je třeba definovat tři základní atributy. Často je možné se setkat se zkratkou *ABC*[35] vycházející z počátečních písmen anglických názvů atributů (viz obrázek 5.3).

Address
Definuje adresu, na které je služba dostupná.

[10] `https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx` [citováno 11. května 2016]

Binding
Definuje způsob, jakým je možné se službou komunikovat.

Contract
Definuje rozhraní (operace), které služba nabízí.



Obrázek 5.3: Tzv. ABC frameworku WCF. Převzato z `https://blogs.msdn.microsoft.com/paolos/2009/11/17/customizing-and-extending-the-biztalk-wcf-adapters/` [citováno 11. května 2016]

Jednou z výhod WCF je dobrá integrace do prostředí platformy .NET, kde je práce se službami v podstatě shodná s prací s lokálními metodami[11]. Infrastrukturu nezbytnou pro komunikaci mezi procesy (obsluha spojení, serializace, přenos, deserializace, atd.) pak zajišťuje WCF framework. Jednou z nevýhod jsou omezení poskytovaného rozhraní (contract). Při vytváření rozhraní služby je třeba brát tato omezení v úvahu a použít odpovídající programové konstrukce. Mezi taková omezení patří například[12] nemožnost přetěžování metod, omezená práce s generikou nebo nutnost specifikovat konkrétní návratový typ metody nebo vlastnosti.

V systému Cerekon využíváme dva způsoby komunikace:

1. **HTTP** pro webové rozhraní
2. **NET.TCP** pro hlavní komunikaci

Jak bylo výše popsáno, způsob komunikace definuje tzv. binding. V našem případě se tedy jedná o `WebHttpBinding`[13] a hlavně `NetTcpBinding`[14]. `NetTcpBinding` poskytuje *„bezpečné, spolehlivé a optimalizované spojení pro komunikaci*

[11] Tomuto popisu ještě více odpovídá jiný způsob komunikace, tzv. .NET remoting.
[12] Některá z těchto omezení je možné různými více či méně triviálními způsoby obejít.
[13] `https://msdn.microsoft.com/cs-cz/library/system.servicemodel.webhttpbinding(v=vs.110).aspx` [citováno 11. května 2016]
[14] `https://msdn.microsoft.com/cs-cz/library/system.servicemodel.nettcpbinding(v=vs.110).aspx` [citováno 11. května 2016]

mezi počítači. Implicitně používá Windows Security pro zabezpečení zpráv a autentifikaci, TCP pro doručování zpráv a binární kódování zpráv.“[15]

Tento způsob komunikace také podporuje koncept tzv. *reliable sessions*[16][17], což je mimo spolehlivost a zabezpečení další z důvodů, pro které jsme se rozhodli pro využití právě NET.TCP komunikace.

Jak již předchozí odstavec napověděl, architektura systému Cerekon využívá konceptu sessions. Jedná se o způsob udržování kontextu (stavu) komunikace mezi klientem a serverem. V praxi se, mimo jiné, používají:

- bezkontextová komunikace - klient si sám obhospodařuje kontext
- přenášení kontextu - s každým požadavkem se přenáší i celý kontext
- kontext je uložen na serveru - udržován v rámci session

Použití sessions umožňuje jednodušší práci v prostředí serveru i klienta. Z pohledu klienta je session přirozený model odpovídající uživatelskému pohledu na práci s aplikací, kde interakce uživatele s aplikací probíhá v rámci jedné session. Díky podpoře sessions ve WCF frameworku není nutné se starat o kontext požadavků explicitně. Vnitřní infrastruktura WCF zajistí, že implementace služby bude klientský požadavek zpracovávat ve správném kontextu.

Inversion of control

Při implementaci systému jsme se drželi paradigmatu „programování proti rozhraním“. Tento přístup říká, že místo konkrétní implementace budeme pro práci s určitou funkcionalitou používat její rozhraní.

Příklad Místo použití implementace:

```
class TestClass
...
TestClass test;
...
test.Method();
```

použít rozhraní:

```
interface ITestInterface
...
ITestInterface test;
...
test.Method();
```

Do proměnné `test` pak můžeme dosadit libovolnou třídu implementující rozhraní `ITestInterface`. V případě použití konkrétní třídy `TestClass` bychom sice

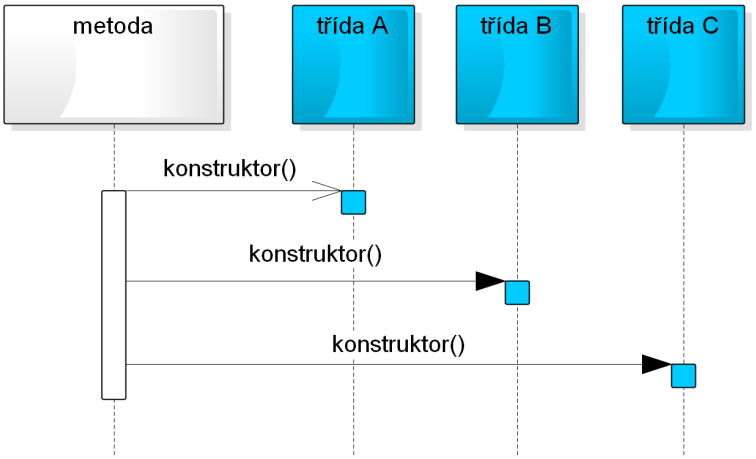
[15] Převzato z dokumentace MSDN: `https://msdn.microsoft.com/cs-cz/library/ms731343(v=vs.110).aspx` [citováno 11. května 2016]

[16] `https://msdn.microsoft.com/en-us/library/ms733136(v=vs.110).aspx` [citováno 11. května 2016]

[17] Dále budeme používat anglický termín `session` běžně používaný i v česky psané literatuře.

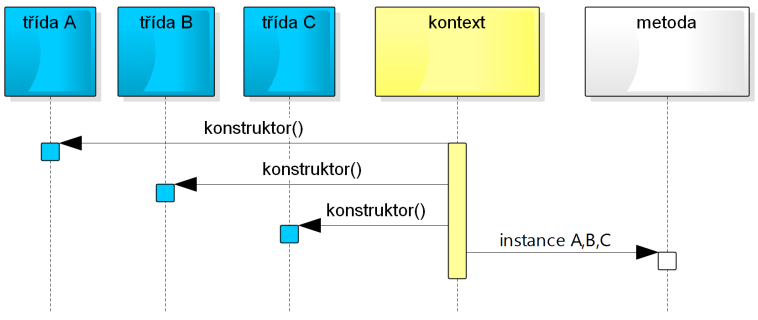
mohli dosadit i libovolného potomka této třídy, avšak v rámci dědičnosti jsme, alespoň v jazyce C#, omezeni na jednoduchou dědičnost. Tedy jedna třída nemůže mít více předků. Oproti tomu může jedna třída implementovat více rozhraní.

Při „klasickém“ toku programu metoda deklaruje svoje lokální proměnné, do kterých přiřazuje potřebné objekty příslušných typů, které sama instanciuje (jako na diagramu 5.4).



Obrázek 5.4: Metoda vytvářející instance tříd A, B a C.

Metoda, potažmo třída, tedy musí znát třídu (konkrétní implementaci), jejíž instanci vytváří a tím se na ní stává závislou. V anglicky psané literatuře se používají termíny „*dependency*“ (závislost) nebo se také říká, že jsou třídy „*coupled*“. S narůstajícím počtem závislostí je zdrojový kód hůře udržovatelný, typicky se zvyšuje také počet duplicit a redundancí, a tím roste i pravděpodobnost chyby v kódu. Princip, někdy také označovaný za návrhový vzor, Inversion of Control (IoC) má za cíl optimalizovat množství závislostí třídy. Místo toku uvedeného výše se použije „*inverzní kontrola*“ - tedy potřebné instance nebude vytvářet daná metoda (třída), ale vzniknou „někde jinde“.



Obrázek 5.5: IoC

Jak ilustruje obrázek 5.5, instance třídy jsou vytvořeny v určitém kontextu a teprve pak jsou předány třídě nebo metodě, která je vyžaduje, resp. vyžaduje rozhraní, která jsou třídami instancí implementována. Tento stav, kdy mezi třídami není závislost, se v anglické literatuře označuje jako „*decoupled*“. Z obrázku dále vyplývá, že závislosti jsou pouze přesunuty do jednoho místa, které je označeno

jako kontext. Právě podoba tohoto kontextu je jednou z odlišností mezi různými implementacemi (typy) IoC.

V serveru systému Cerekon jsme používali techniku zvanou „dependency injection“. V rámci tohoto přístupu je dříve zmíněný kontext tvořen tzv. „kontejnerem závislostí“(dependency container). Kontejner poskytuje rozhraní pro registraci rozhraní a jejich příslušných implementací. Pokud třída potřebuje instanci typu implementujícího požadované rozhraní, kontejner vrátí již dříve vytvořenou instanci, nebo vytvoří novou. Pokud jsou k vytvoření instance potřebné další instance, kontejner postupuje rekurzivně. Poskytuje tak velmi silný mechanizmus pro automatické vytváření instancí tříd, aniž by bylo nutné se zabývat vytvářením nebo získáváním potřebných objektů.

V našem systému používáme kontejner z knihovny Castle Windsor[18]. Tento kontejner navíc podporuje řadu dalších funkcí (oproti jiným kontejnerům, např. Unity[19]), které jsme mohli při vývoji použít.

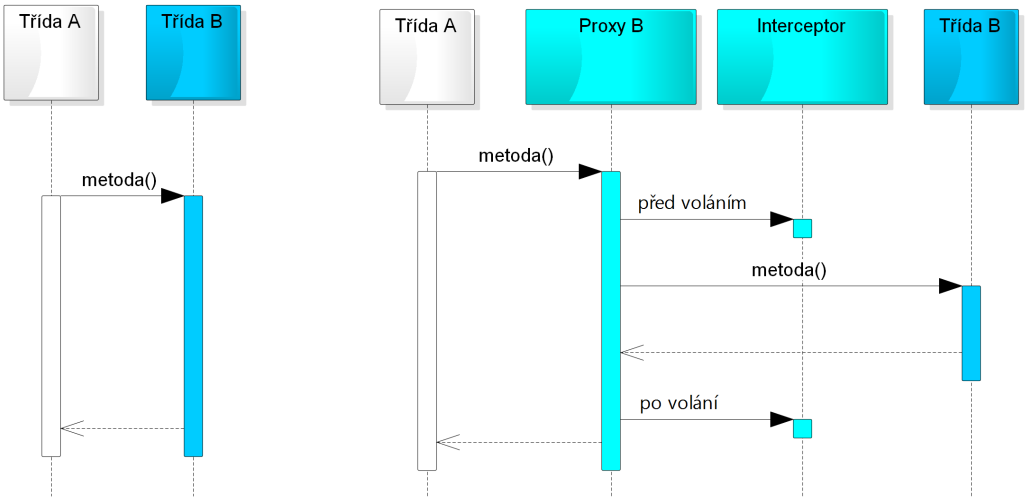
Naším cílem není podrobně popsat všechny techniky a vlastnosti dependency injection, kterým se věnuje např. uznávaná autorita Martin Fowler[36]. Chtěli jsme alespoň nastínit a zdůvodnit principy, kterých jsme se při vývoji AS drželi.

Aspect oriented programming

Další technikou, kterou jsme při vývoji využili, je aspektově orientované programování (aspect-oriented programming, AOP). Hlavní myšlenkou tohoto přístupu je, že různé části části programu mají nebo mohou mít různé aspekty. Aspekty de facto znamenají použití téže části programu v různých kontextech. V objektově orientovaných jazycích se setkáváme s implementací AOP pomocí tzv. „*interceptorů*“. Použitím různých interceptorů se pak odlišují různé aspekty. Interceptor, jak název napovídá, z pohledu uživatele (v tomto případě je uživatelem přímo programátor) zachytí volání metody a umožní před nebo po jeho vykonání spustit další kód.

[18] `http://www.castleproject.org/` [citováno 11. května 2016]

[19] `https://msdn.microsoft.com/en-us/library/ff647202.aspx` [citováno 11. května 2016]



Obrázek 5.6: Vlevo: standardní volání metody. Vpravo: Volání metody pomocí proxy-třídy. Interceptor může vykonávat další kód před i po spuštění volané metody.

Podpora AOP je implementována nejčastěji těmito způsoby:

Při překladu
: Na základě statické analýzy kódu je při překladu vygenerována potřebná infrastruktura definovaná aspekty[20].

Za běhu
: Za běhu programu nejsou vytvářeny instance definovaného typu, ale instance tzv. proxy-třídy se stejným rozhraním. Metody této proxy-třídy pak obsluhují chování definované aspekty (viz obrázek 5.6).

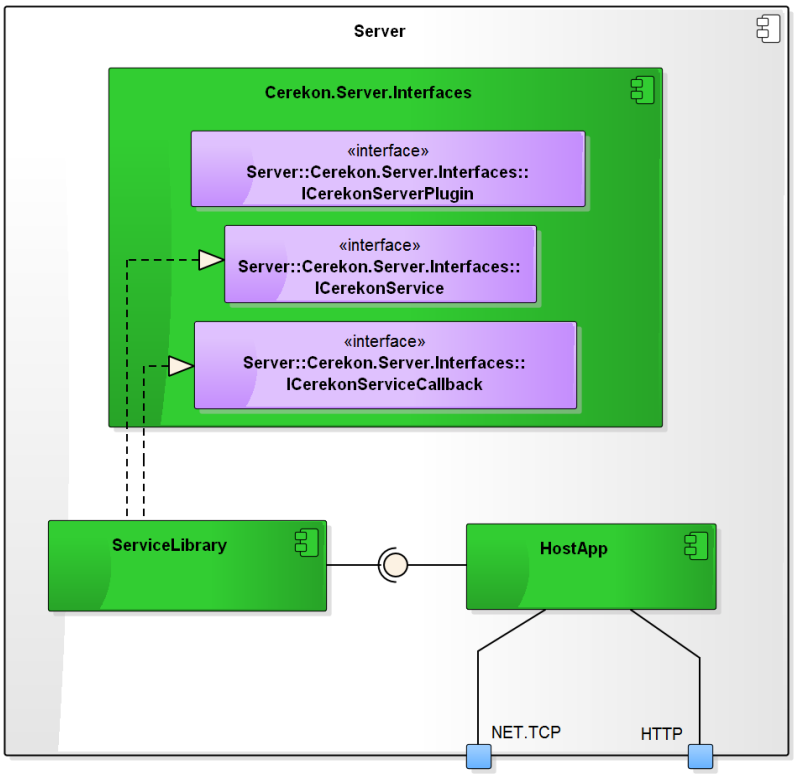
Stejně jako v předchozím případě jsme i pro tyto účely využili knihovnu Castle.Windsor, která poskytuje podporu AOP pomocí proxy-tříd. Zde se opět vracíme k použití dependency injection - pokud si třída vyžádá instanci implementující dané rozhraní, dependency kontejner jí vrátí instanci proxy-třídy implementující potřebné rozhraní.

Typickými scénáři použití AOP jsou logování událostí nebo autorizace požadavků. Právě pro tyto účely jsme AOP použili v systému Cerekon. Další informace o AOP v prostředí .NET je možné získat v [37] nebo původním článku [38].

5.3.2 Architektura

Následující obrázek ilustruje jednotlivé logické komponenty, ze kterých se server skládá:

[20]Příkladem může být např. Postsharp: `https://www.postsharp.net/aop.net` [citováno 11. května 2016]



Obrázek 5.7: Komponenty aplikačního serveru.

5.3.3 Cerekon.Server.Interfaces

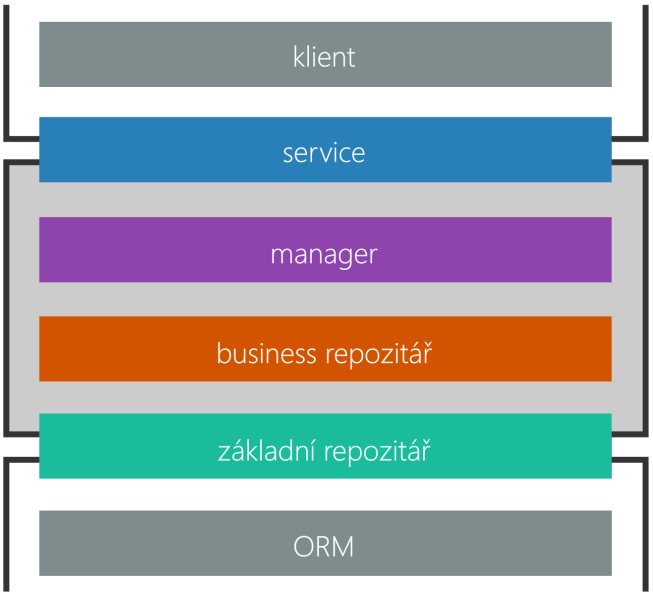
Tato část obsahuje definice rozhraní potřebných pro komunikaci s aplikačním serverem. Tedy rozhraní služby serveru Cerekon, rozhraní callbacku služby[21] a rozhraní vyžadované od serverové části každého rozšiřujícího modulu, tzv. pluginu.

5.3.4 ServiceLibrary

V této knihovně je uložena implementace logiky aplikačního serveru. Většina funkcionality aplikačního serveru je implementována pomocí struktur barevně vyznačených na obrázku 5.8. Opět se na tomto místě vracíme k rozdělení architektury do vrstev.

Na obrázku je stejně jako v logické hierarchii vrstva ORM, kterou však nezařazujeme do standardní architektury serveru, neboť AS je od ORM odstíněn pomocí návrhového vzoru repozitáře (viz 5.2). Z tohoto důvodu považujeme za nejnižší vrstvu serveru právě repozitář, přesněji řečeno tzv. **základní repozitář**. Základní repozitář přesně odpovídá zmíněnému návrhovému vzoru, kdy zodpovídá za manipulaci s právě jedním datovým typem. V hierarchii výše umístěný **business repozitář** oproti základnímu repozitáři poskytuje rozhraní pro více typů. Tato „tenká vrstva“ zastřešuje základní repozitáře a poskytuje rozhraní pro přístup k datům potřebné pro vyšší vrstvy serveru. Další vrstvou v hierarchii je tzv. **manager**. Jedná se implementaci návrhového vzoru *business object*,

[21] Význam bude upřesněn dále v implementaci serveru.



Obrázek 5.8: Vrstvy aplikačního serveru.

avšak záleží na použité definici vzoru. Manager implementuje aplikační logiku a odděluje ji od vrstvy přistupující k datům i od vrstvy klientského rozhraní. Nejvyšší vrstvou AS je vrstva **služby**, jejíž rozhraní odpovídá požadavkům na služby přístupné pomocí WCF. Metody služby neobsahují typicky žádnou logiku, pouze přesměrovávají požadavky na příslušné managery.

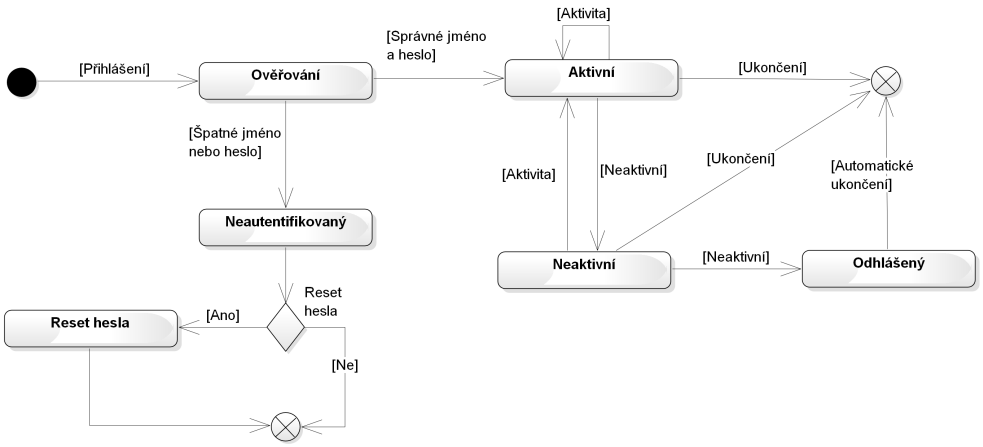
V dalších odstavcích popíšeme některé z vnitřních mechanizmů serveru.

Uživatelská session

Session pro uživatele začíná jeho pokusem o přihlášení pomocí uživatelského jména a hesla. Stejným požadavkem je vytvořena i session frameworku WCF. WCF umožňuje provést autentifikaci ještě před vytvořením session, my jsme se však rozhodli postupovat odlišně. Session je vytvořena ještě před autentifikací a autorizací. Autentifikace je pak prvním krokem v nově vytvořené session. Tento přístup umožňuje používat autentifikaci jako standardní funkci v rámci aplikačního serveru. V případě neúspěšné autentifikace uživatel rozhodne, má-li se jeho heslo resetovat (nahradit náhodně generovaným řetězcem). Pokud ano, nové heslo je zasláno na vyplněný e-mail. V obou případech session končí.

Pokud je autentifikace úspěšná, session dále pokračuje a v rámci jejího kontextu probíhá vyřizovaní požadavků přicházejících od klienta. V rámci WCF je možné pro session nastavit různé druhy timeoutů[22]. Jedním z nich je i timeout na neaktivitu uživatele. Tedy pokud uživatel po určitou dobu nevykáže žádnou aktivitu, je jeho session ukončena. Nastavení takového timeoutu na konečnou hodnotu je racionální z hlediska bezpečnosti (např. zapomenutá spuštěná aplikace s přihlášeným uživatelem) i z hlediska výkonu (každá session spotřebovává zdroje serveru). Pouhé využití timeoutů poskytnutých WCF frameworkem je dle našeho názoru uživatelsky nepřívětivé a proto jsme implementovali jakousi nadstavbu na

[22]Přestože český překlad „časový limit“ se v praxi využívá, budeme nadále používat anglický výraz „timeout“ běžně užívaný i v česky psané literatuře.



Obrázek 5.9: Diagram stavů session.

aplikační úrovni. Tato nadstavba spočívá ve využití dvou časovačů. První časovač slouží k odpočítávání doby, po kterou klient nevykázal potřebnou aktivitu. Po uplynutí tohoto času je klient informován, že je dlouhou dobu neaktivní a jeho session by mohla být ukončena. V tuto chvíli se spouští odpočet druhého časovače. Pokud ani do vypršení tohoto druhého časovače není klient aktivní, je mu o tom zaslána informace a session je posléze ukončena. Diagram stavů session ilustruje obrázek 5.9.

Aby bylo možné takový mechanizmus efektivně implementovat, je potřeba použít takový způsob spojení, kde server může aktivně iniciovat komunikaci s klienty. Spojení, které tuto komunikaci umožňuje, se označuje jako **duplexní**. Podpora duplexního spojení je dalším z důvodů komunikace přes net.tcp (viz 5.3.1) zmíněný dříve v této kapitole. Ve WCF je duplexní komunikace (ze serveru ke klientovi) implementována pomocí tzv. „callbacku“. Potřebná rozhraní pro callbacky jsou definována v rámci Cerekon.Server.Interfaces. Prostřednictvím těchto rozhraní pak probíhá komunikace ze serveru na klienty.

Vlastní mechanizmus řízení timeoutů nám poskytl možnost aktivně udržovat session mezi klientem a serverem pomocí tzv. „heartbeat zpráv“. Framework WCF sice tento princip interně využívá[23], v praxi se nicméně ukázalo, že vnitřní implementace nemusí vždy fungovat korektně. V takových případech pak dochází k předčasnému ukončení session. Vlastní udržování session společně s vlastní implementací timeoutů nám umožnilo tyto problémy jednoduše obejít.

Autorizace

Z požadavků (viz 3.1.1) vyplývá, že jsme museli řešit otázku autorizace uživatelů. Rozhodli jsme se využít v dnešní době často implementovaný mechanizmus založený na rolích a oprávněních. V následující tabulce vysvětlíme vztah mezi uživatelem, rolí a oprávněním:

[23]Pomocí tzv. „infrastructure messages“.

Entita	Popis
Oprávnění	Oprávnění jsou pevně definována v systému a nelze je uživatelsky měnit. Oprávnění odpovídá určité akci provedené v systému. Každé oprávnění je identifikované pomocí GUID.
Role	Role agregují oprávnění. Tedy jedna role má přiřazena více oprávnění a zároveň jedno oprávnění může být přiřazeno více rolím. Role je možné definovat hierarchicky, obdobně jako dědičnost tříd, kde podřízená role přebírá všechna oprávnění nadřazené role.
Uživatel	Uživatel může mít více rolí a jednu roli může mít více uživatelů.

Tabulka 5.1: Rozdělení oprávnění, rolí a uživatelů.

Autorizace je u příslušné metody specifikována formou aspektu AOP, kde jsou vyjmenována potřebná oprávnění. Pro přehlednost jsme vytvořili strukturu a definovali pravidla pro vytváření oprávnění. Pro definici oprávnění jsou určeny třídy[24] implementující rozhraní `IPermissionsContainer`. Toto rozhraní poskytuje jedinou metodu `GetPermissions`, která vrací seznam všech oprávnění definovaných v rámci třídy. Samotné oprávnění je pak definováno jako veřejná konstanta typu `Guid`. Díky tomuto přístupu je možné v rámci kódu pracovat s názvy konstant místo přímo s GUID oprávnění. Navíc je tak možné jednoduše získat seznam všech oprávnění definovaných v rámci celého systému.

Rozhraní pro web

Pro snadnější údržbu a správu systému jsme implementovali rozhraní aplikačního serveru, které je přístupné v podobě webové služby pomocí protokolu HTTP. Pomocí tohoto rozhraní je možné získat informace o běžící instanci AS Cerekon. Tři nejdůležitější funkce služby jsou zaznamenány v tabulce:

Funkce	Popis
ServerInfo	Poskytuje základní parametry spuštěné instance serveru (např. platforma, verze, stav paměti, etc.)
Permissions	Vrací seznam všech definovaných oprávnění jako seznam dvojic jméno - GUID.
ServerMode	Vrací identifikátor typu instance serveru (vývojová, testovací nebo produkční).

Tabulka 5.2: Funkce webové služby.

Poslední funkcí, kterou v souvislosti s AS a webovým rozhraním zmíníme, je funkce „notifikace session“. Jedná se o funkčnost, kdy jsou vytvářeny notifikace navázané na některé akce v serveru. Takové akce jsou:

- vytvoření a ukončení session

[24] V jazyce C# není možné definovat globální pole nebo proměnné, vše musí být definováno v rámci nějaké třídy.

- aktivita klienta v rámci session
- heartbeat zpráva od klienta
- uplynutí timeoutu neaktivity klienta

V naší implementaci jsou notifikace dále propagovány do webového rozhraní. Naším cílem bylo v tomto případě zobrazovat informace o session ve webovém rozhraní bez nutnosti obnovovat stránku. Přestože se jedná o požadavek patřící spíše do kapitoly věnované klientské části, je nezbytné vytvoření podpory na straně serveru. Pro zpřístupnění notifikací ve webovém prostředí jsme použili technologii **WebSockets**[25]. Tato technologie umožňuje ponechat otevřené spojení mezi webovým klientem a webovým serverem. Jelikož není podpora WebSocekts ve WCF úplně přímočará, použili jsme jako prostředníka jednoduchý webový server Node.js[26]. Pro komunikaci mezi AS a Node.js slouží knihovna SocketIoClientDotNet[27]. Protože se nejdená o zásadní funkci serveru, nebudeme se zaobírat dalšími detaily.

5.3.5 CerekonServiceHostApp

Poslední součástí aplikačního serveru Cerekon je jednoduchá konzolová aplikace, která zpřístupňuje, resp. tzv. „hostuje“, výše popsanou službu. Pro WCF služby se používají nejčastěji tři typy hostování služeb:

V IIS
: Služba běží v rámci webového serveru Microsoft Internet Information Services[28].

Windows Service
: Služba WCF je spuštěna jako služba systému Windows (Windows Service).

Self hosting
: Služba WCF je spuštěna v rámci aplikace.

Pro vlastní aplikaci, tedy self-hosting, jsme se rozhodli pro jednoduchou instalaci a spravovatelnost. Aplikace samotná pouze ověří, zda má uživatel administrátorská oprávnění, která jsou potřebná ke spuštění služby, načte konfigurační soubory a spustí WCF službu Cerekon společně s Node.js webovým serverem. Výhodou konzolové aplikace je také jednoduché čtení výstupů.

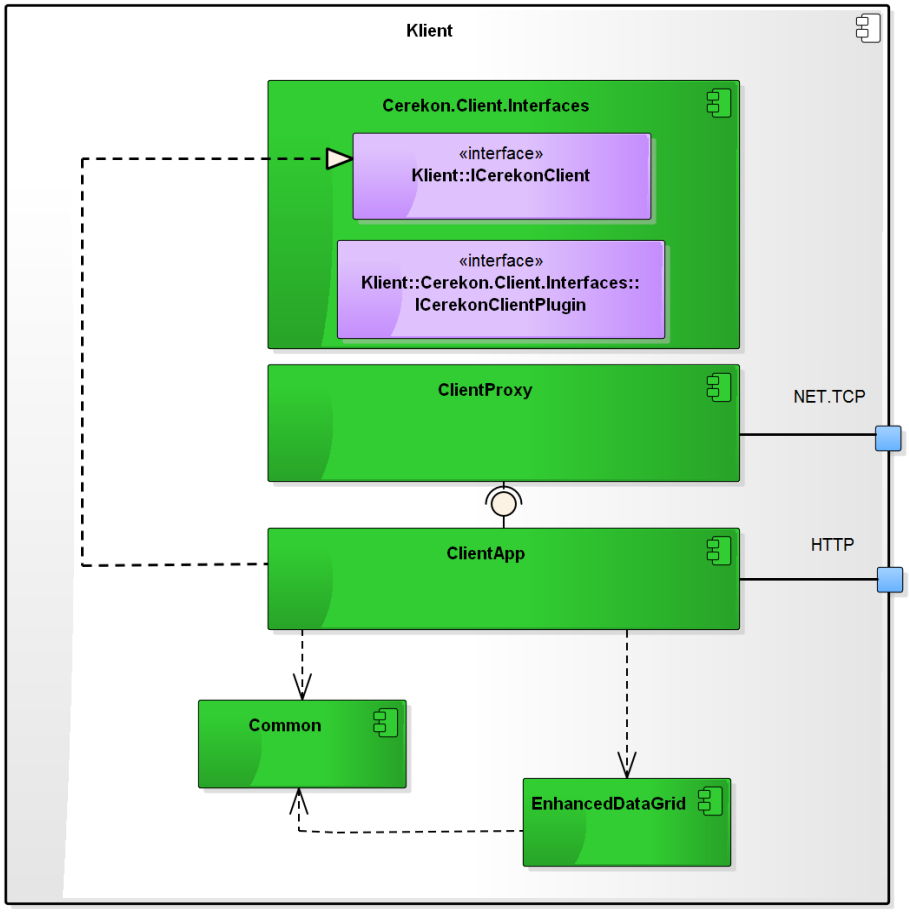
[25] `https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API` [citováno 11. května 2016]
[26] `https://nodejs.org/en/` [citováno 11. května 2016]
[27] `https://github.com/Quobject/SocketIoClientDotNet` [citováno 11. května 2016]
[28] `http://www.iis.net` [citováno 11. května 2016]

6. Klient

Podle požadavků (viz 3.1.2) má být klientská aplikace[1] implementována v desktopovém prostředí systému Windows. Stejně jako v případě AS jsme se i v tomto případě rozhodli pro implementaci v jazyce C#. Pro tvorbu uživatelského rozhraní jsme využili framework Windows Presentation Foundation (WPF)[2]. WPF je v dnešní době na platformě .NET a Windows považováno v podstatě za standard díky mnoha pokročilým funkcím a možnostem definování prakticky libovolného designu aplikací. Na úvod představíme základní rozdělení a architekturu klientské části systému.

6.1 Architektura

Níže uvedený diagram znázorňuje rozdělení klienta na jednotlivé komponenty:



Obrázek 6.1: Komponenty klientské části systému.

[1] V požadavcích je specifikován obecně celý systém, nicméně z pohledu uživatele je nejdůležitější právě klientská část systému.

[2] `https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx` [citováno 11. května 2016]

6.1.1 Model-view-viewmodel

Při vývoji klienta ve frameworku WPF jsme volili mezi dvěma přístupy k implementaci uživatelského rozhraní (UI):

Pomocí událostí
: Aplikace je řízena událostmi, které vyvolává uživatelské rozhraní. Stav aplikace se mění při obsluze události.

Pomocí data bindingu
: Data binding[3][4] je způsob propojení nejčastěji dvou datových položek. Jednou položkou je typicky prvek UI, druhou položkou pak pole z datového modelu. V podstatě se jedná o implementaci vzoru Vydavatel-Předplatitel[39] (publisher-subscriber)[5]

Zvolili jsme použití data bindingu[6], které umožňuje lepší oddělení UI od zbytku prezentační vrstvy aplikace, což je i jedním z cílů vzoru model-view-viewmodel (MVVM)[7]. Tento vzor rozděluje prezentační vrstvu aplikace[8] do dalších třech vrstev:

View
: View obsahuje prvky UI, např. okna, textová pole nebo tlačítka.

Viewmodel
: Viewmodel je model dat potřebných pro UI. Např. řetězec pro textové pole, pole objektů pro rozbalovací seznam, apod.

Model
: Model je vrstva standardního objektového modelu typicky vycházejícího z doménového modelu aplikace.

[3] Jelikož se jedná o běžně rozšířený termín, nebudeme jej překládat.

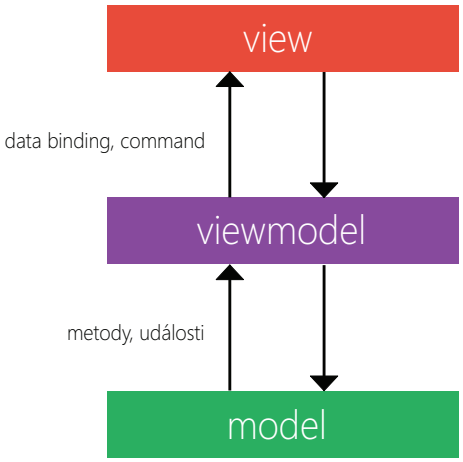
[4] `https://msdn.microsoft.com/en-us/library/ms752347(v=vs.100).aspx` [citováno 11. května 2016]

[5] `https://msdn.microsoft.com/en-us/library/ff649664.aspx` [citováno 11. května 2016]

[6] Až na některé výjimky, kde WPF data binding použít nedovoluje.

[7] `https://msdn.microsoft.com/cs-cz/library/hh848246.aspx` [citováno 11. května 2016]

[8] V praxi se můžeme setkat s tvrzením, že je aplikace implementovaná pomocí vzoru MVVM. Zde je však nutno mít na paměti, že MVVM je vzorem pouze pro prezentační vrstvu aplikace.

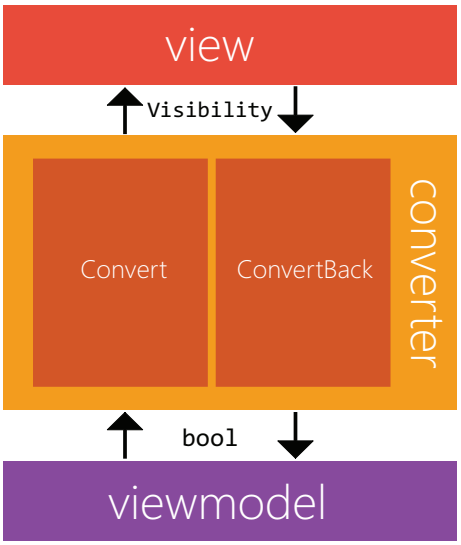


Obrázek 6.2: Prezentační vrstva podle vzoru MVVM.

Jak ilustruje obrázek 6.2, komunikace mezi view a viewmodelem je zprostředkována právě prostřednictvím data bindingu. Bez velké újmy na obecnosti dále budeme o MVVM mluvit v kontextu WPF. Na straně view je svým jménem specifikován zdroj (source) bindingu, kterým je nějaká veřejná vlastnost (public property) viewmodelu. Až na názvy je tak view nezávislé na modelu a obráceně. Je tedy možné pro jedno view (uživatelské rozhraní) používat různé viewmodely a měnit tak logiku aplikace. V rámci definice data bindingu je možné specifikovat i další parametry. Jedním z nejpoužívanějších parametrů jsou tzv. „konvertory“[9]. Někdy se konvertory dokonce označují za samostatnou vrstvu. Konvertor umožňuje efektivně řešit rozdílnost reprezentace jednoho atributu ve view a viewmodelu. Uveďme příklad, který se v mnoha aplikacích, jakož i naší implementaci vyskytuje:

Příklad Chceme zobrazit tlačítko pouze v případě, že je formulář vyplněn správně. Viewmodel bude obsahovat pole odpovídající formuláři a vlastnost `IsReady` typu `Boolean`. Pokud je formulář správně vyplněn, nastavíme hodnotu `IsReady = true`, přičemž způsobem provedení validace se v tuto chvíli nebudeme zabývat. Na straně view mějme tlačítko, které má vlastnost `Visibility`, kterou chceme data bindingem propojit s vlastností `IsReady` viewmodelu. Vlastnost `Visibility` je však typu `System.Windows.Visibility` a tedy musíme použít konvertor, který bude pro hodnoty `Boolean` vracet hodnoty ze `System.Windows.Visibility`. Po nastavení vlastnosti `IsReady` se nová hodnota předá konvertoru a pokud je konverze úspěšná, návratová hodnota se dále propaguje do vlastnosti `Visibility`.

[9] `https://msdn.microsoft.com/cs-cz/library/system.windows.data.ivalueconverter(v=vs.110).aspx` [citováno 11. května 2016]



Obrázek 6.3: Použití konvertoru v data bindingu.

Na obrázku 6.2 je vedle data bindingu explicitně uvedený i „command“. Ve WPF je do určité míry implementovaný návrhový vzor „command pattern“. Command, tedy příkaz, je transformace určitého algoritmu (chování) do datové struktury (objektu). Zřejmě lze tedy s chováním a tím i funkčností aplikace manipulovat jako s objekty, například použít je jako zdroj pro data binding. Jedná se z určitého pohledu o MVVM ekvivalent událostí.

Obrázek 6.4: Command pattern. Vlevo: použití událostí - k události `Click` je registrována metoda `OnClick` obsahující výkonný kód. Vpravo: použití command patternu - k vlastnosti `ClickCommand` je data bindingem připojena vlastnost `DoWorkCommand` implementující metodu `Execute` obsahující výkonný kód.

6.2 Common

Použití MVVM a jiných návrhových vzorů znamená používat na více místech aplikace obdobné nebo totožné programové konstrukce. Abychom se vyhnuli zbytečnému duplikování kódu[10], vyčlenili jsme některé obecné prvky do separátní knihovny, kterou jsme nazvali `Common`. V tabulce uvedeme stručný přehled některých položek knihovny Common:

[10]Používá se též termín „pravidlo DRY“ z anglického **D**on’t **R**epeat **Y**ourself, tedy „neopakuj se“.

Název	Popis
ViewModelBase	Společný předek pro další viewmodely.
Proxy	Pomocná třída umožňující nepřímé provázání pomocí data bindingu.
ModelsCreator	Mechanizmus pro dynamické vytváření tříd (bude popsáno později).
ValidationRules	Validátory používané pro kontrolu dat.
ValueConverters	Konvertory pro data binding.

Tabulka 6.1: Přehled některých funkcí z knihovny Common.

6.2.1 Enhanced datagrid

Jedním z požadavků kladených na aplikaci je i možnost jednoduše provádět základní analýzu uložených dat. Data mají v přirozeném kontextu i v databázi podobu tabulky a proto jsme řešili požadavek na základní analytickou funkčnost právě na úrovni tabulek. Naším základním cílem bylo pro tabulky implementovat tuto funkcionalitu:

- Možnost filtrování dat v tabulce podle atributů zobrazených v tabulce.
- Možnost zobrazit základní statistické údaje pro každý sloupec v tabulce.
- Možnost vytvářet agregace na základě uživatelem vybraných podmnožin sloupců.

Z pohledu uživatele i UI je logická tabulka typicky reprezentována grafickým prvkem, který má také vzhled a chování tabulky. Takový prvek budeme dále označovat pomocí často užívaného výrazu **grid**. Gridem se ve frameworku WPF obecně myslí komponenta **DataGrid**[11]. DataGrid disponuje poměrně bohatým API, které umožňuje např. řazení sloupců i základní podporu pro vyhledávání. Našim požadavkům však standardní API nedostačovalo a proto jsme museli přistoupit k vlastní implementaci.

Implementovat celý grid od začátku by bylo zbytečné, postupovali jsme tedy tak, abychom mohli využít standardní DataGrid z WPF. Náš postup bychom mohli označit de facto za implementaci vzoru „dekorátor“(decorator pattern), kdy deklaraci instance DataGridu, resp. příslušných sloupců, rozšiřujeme o námi definované atributy potřebné pro implementaci požadovaných vlastností.

Výše popsanou funkcionalitu souhrnně nazýváme **Enhanced Data Grid** (EDG), od čehož je odvozen i název této sekce. Vnitřní strukturu EDG ilustruje následující obrázek:

[11] `https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid(v=vs.110).aspx` [citováno 11. května 2016]

Obrázek 6.5: Komponenty EnhancedDataGridu optimalizované pro vzor MVVM.

Common

Knihovna Common obsahuje struktury používané v dalších částech EDG. Mezi nejdůležitější patří výčtové typy reprezentující možné stavy gridu a rozšířená implementace command patternu, umožňující definovat akci přiřazenou nejen pro samotný příkaz, ale i operace provedené před i po provedení hlavní akce a dále možnost definice transformace dat. Další důležitou součástí jsou vlastnosti implementované pomocí tzv. „attached properties“[12], kterými se dekorují sloupce gridu pro podporu rozšířené funkčnosti. Mezi takové atributy patří např. typ sloupce, odpovídající atribut modelu nebo typ použitého konvertoru.

Filters

Druhá knihovna je věnována filtrům, tedy implementaci omezení dat na základě uživatelem definované selekce. Na nejvyšší úrovni se filtr skládá z jediného zastřešujícího objektu filtru, který pak obsahuje žádný, jeden nebo více konkrétních atomických filtrů definovaných nad sloupci. V základní implementaci jsme připravili následující možnosti filtrování dat:

[12] `https://msdn.microsoft.com/en-us/library/ms749011(v=vs.100).aspx` [citováno 11. května 2016]

Typ sloupce	Typ filtru
Neurčený	Žádný
Datum	Intervalový
Celé číslo	Intervalový
Desetinné číslo	Intervalový
Text	Vyhledávání na začátek, konec, přesnou shodu
Logická hodnota	Ano/ne
Každý filtr má navíc možnost „Nevyplněno“.	

Tabulka 6.2: Přehled typů atomických filtrů.

Filtry je možné kombinovat. V případě, že jsou filtry definovány nad různými sloupci, výsledkem je jejich konjunkce, pokud nad stejným sloupcem, výsledkem je jejich disjunkce, přičemž selekce nad jedním atributem jsou seskupeny.

Příklad Filtr: Atribut A mezi 1 a 3, B mezi 0 a 10, A nevyplněno, C začíná na „c“, B nevyplněno, A menší než -5. Selekce bude vypadat obdobně:

$$((((A \in < 1{,}3 > \vee A = \omega) \vee (A \leq -5)) \wedge (B \in < 0{,}10 > \vee B = \omega)) \wedge (C\ like\ '\%c'))$$

Filtry jsou navrženy primárně pro práci s aplikačním serverem, resp. persistentním úložištěm. Skrz svoje API filtr zpřístupní selekci, kterou je možné propagovat do nižších vrstev systému a filtrování provádět až na úrovni úložiště (databáze), což je efektivnější, než pozdější filtrování na klientovi.

Statistics

Tato část obsahuje funkce podpory základních statistik nad daty. Oproti filtrům jsou statistiky vyhodnocovány na klientovi a to s použitím aktuálně načtených, tedy i potenciálně vyfiltrovaných dat. Statistiky zde dělíme do dvou typů:

- jednoduchá statistika
- rozšířená statistika

Jednoduchou statistiku můžeme definovat jako množinu statistických údajů, kde každý údaj patří právě jednomu sloupci a jejich počet je shora omezen počtem sloupců. Jednoduše řečeno, každý sloupec může mít v jeden okamžik nejvýše jednu jednoduchou statistiku.

Jednoduchá i rozšířená statistika, které se budeme věnovat později, využívá tzv. „agregátory“. Základní funkcí agregátoru je zřejmě provádění agregací, tedy na vstupu dostane kolekci hodnot a výstupem je jediná hodnota. Implementovali jsme tyto základní agregátory:

- Počet
- Počet unikátních
- Minimum
- Maximum

- Suma
- Aritmetický průměr
- Počet logických hodnot ano/ne/nevyplněno

Jednoduchá statistika je pak výsledkem aplikace jednoho agregátoru na daný sloupec.

Rozšířená statistika pracuje vždy s celou tabulkou (relací)[13], kromě sloupců, které mají typ obsahu nedefinovaný nebo definovaný jako „žádný“. Rozšířenou statistiku jsme implementovali obdobně jako operátor `GROUP BY`[14] v jazyce SQL. V tomto kontextu rozdělujeme sloupce do dvou skupin: **seskupující** a **agregované**, přičemž sloupec účastnící se statistiky patří do právě jedné z těchto dvou skupin. Seskupující sloupce slouží ke stanovení domény výsledku rozšířené statistiky. Definujme doménu formálněji:

Definice 4 (Doména rozšířené statistiky)**.** *Doménou rozšířené statistiky je kartézský součin aktuálních domén atributů reprezentovaných seskupujícími sloupci. Buď C_G množina všech seskupujících sloupců rozšířené statistiky S. Pak:*

$$dom(S) = \underset{c \in C_G}{\times} adom(c).$$

Výsledkem rozšířené statistiky je tabulka, resp. relace. Formálněji zapsáno:

Definice 5 (Rozšířená statistika)**.** *Mějme relaci $R(A)$, kde $A = \{C_{G1}, ..., C_{Gn}, A_1, ...A_m\}$ a $C_G = \{C_{G1},...,C_{Gn}\}$ je množina seskupujících atributů. Dále buď $\Gamma = \{\gamma_1,..\gamma_m\}$ množina agregátorů. Pak rozšířená statistika je relace $S(A')$, kde $A' = \{C_{G1},...,C_{Gn},A'_1,...A'_m\}$ a C_G je klíčem relace S. Označme formuli $\phi = (C_{G1} = c_{g1} \wedge ... \wedge C_{Gn} = c_{gn})$. Pro prvek relace $S^*(A')$ s klíčem $\{c_{g1},...,c_{gn}\}$ platí:*

$$S^*(\phi) = (c_{g1},...,c_{gn},\gamma_1(R^*(\phi)[A_1]),...,\gamma_m(R^*(\phi)[A_m]))$$

Výsledná relace S^* má právě $|dom(S)|$ prvků.

Dále zavádíme předpoklad, aby na doménách seskupujících atributů bylo definováno uspořádání. Tohoto předpokladu využíváme v algoritmu výpočtu rozšířené statistiky.

Z implementačního hlediska je důležitý fakt, že domény atributů výsledné relace mohou být odlišné od původních domén, řečeno jazykem algebry nemusí být množina hodnot atributu uzavřená na operaci agregace. Nejjednodušším příkladem je agregátor aritmetického průměru na celých číslech. Při implementaci je to obzvláště důležité, neboť není obecně možné pro výsledek agregace použít datový typ vstupu. Z tohoto důvodu jsme implementovali funkcionalitu pojmenovanou jako „ModelsCreator“ (viz 6.1). Využíváme zde předpokladu, že výsledek libovolné agregace je možné zobrazit jako řetězec znaků. ModelsCreator umožňuje dynamicky za běhu aplikace vytvořit novou třídu, která bude mít stejné vlastnosti jako předložená třída s tím rozdílem, že datovým typem vlastností bude řetězec.

[13] V této části můžeme pojmy tabulky a relace a na ně vázané termíny (např. sloupec a atribut) považovat za ekvivalentní.

[14] `https://msdn.microsoft.com/en-us/library/ms177673.aspx` [citováno 11. května 2016]

Algoritmus 1 Algoritmus pro výpočet rozšířené statistiky.

```
1: function STATISTIKA(data, seskupujiciAtributy,agregatory)
2:     seřaď data podle seskupujících atributů
3:     vysledek ← []
4:     seznam ← [data[0]]
5:     for I := 1 .. Length(data)-1 do
6:         if hodnoty seskupujících atributů se pro data[I] a data[I-1] liší
   then
7:             novyPrvek ← vytvořNovýPrvek(data[I-1],
8:                     seskupujiciAtributy)
9:             nastavPrvek(novyPrvek, seznam, seskupujiciAtributy,
10:                     agregatory)
11:            přidej novyPrvek do vysledku
12:            seznam ← [data[i]]
13:        elsepřidej data[I] do seznamu
14:        end if
15:    end for
16:    novyPrvek ← vytvořNovýPrvek(data[I], seskupujiciAtributy)
17:    nastavPrvek(novyPrvek,seznam,seskupujiciAtributy,agregatory)
18:    přidej novyPrvek do vysledku
       return vysledek
19: end function
```

Algoritmus 2 Vytvoření nového prvku rozšířené statistiky.

```
1: function VYTVOŘNOVÝPRVEK(data, seskupujiciAtributy)
2:     vysledek ← vytvořModel(data)
3:     for all seskupujiciAtribut in seskupujiciAtributy do
4:         vysledek.seskupujiciAtribut := data.seskupujiciAtribut
5:     end for
       return vysledek
6: end function
```

Algoritmus 3 Nastavení hodnot prvku výsledku.

```
1: function NASTAVPRVEK(novyPrvek, seznam, seskupujiciAtributy, agre-
   gatory)
2:     atributy ← atributy novyPrvek-seskupujiciAtributy
3:     for I := 0 .. Length(atributy)-1 do
4:         novyPrvek[atributy[I]] ← agregatory[I].agreguj(seznam,
5:                     atributy[I])
6:     end for
7: end function
```

Takový postup, tedy dynamické vytváření typů a jejich načtení do aplikační domény, nám umožňuje mechanizmus reflection z .NET frameworku[15]. Tento postup je také výhodný při použití MVVM. Stačí si uvědomit, že data binding je definován pomocí názvů vlastností, které náš postup zachovává. Pokud je tedy cíl data bindingu zkonstruován také dostatečně obecně, je možné za běhu měnit původní model za instance nově vytvořené třídy při zachování funkčnosti data bindingu.

Grid

Poslední část s názvem Grid obsahuje komponentu EnhancedDataGrid pro uživatelské rozhraní, tzv. „user control“[16] a viewmodel implementující potřebné vlastnosti. Z pohledu UI je EDG rozdělen do tří částí:

- nastavení filtrů a rozšířené statistiky
- grid zobrazující data
- řádek obsahující výsledky jednoduché statistiky

Abychom minimalizovali zásah do uživatelského rozhraní, ovládání EDG probíhá pomocí kontextového menu vyvolaného kliknutím pravého tlačítka myši na záhlaví sloupce. Kontextové menu nabízí přidání filtru na sloupec, přidání sloupce mezi seskupující sloupce rozšířené statistiky, export obsahu gridu do souboru a přidání nebo odebrání jednoduché statistiky. Obsah kontextové nabídky se může lišit podle aktuálního stavu gridu, stavu sloupce a nebo datového typu sloupce. Možné stavy a přechody zachycuje diagram stavů na obrázku 6.6.

V předchozím odstavci jsme jako jednu z položek menu uvedli export do souboru. EDG umožňuje uložit svůj aktuální stav (tedy i po aplikaci filtrů nebo rozšířené statistiky) do souboru ve formátu Office Open Xml, který je možné později otevřít např. v aplikaci Microsoft Excel. Pro vytváření souborů exportů jsme použili knihovnu **EPPlus**[17] umožňující jednoduchou práci s tabulkami přímo v prostředí jazyka C# a navíc nevyžaduje, aby byla lokálně dostupná instalace Microsoft Excel.

6.3 Cerekon.Client.Interfaces

Obdobně jako v případě serverové části jsme i v klientské části oddělili samotnou implementaci aplikace od požadovaného rozhraní. Na tomto místě je také definováno rozhraní pro klientskou část rozšiřujících modulů.

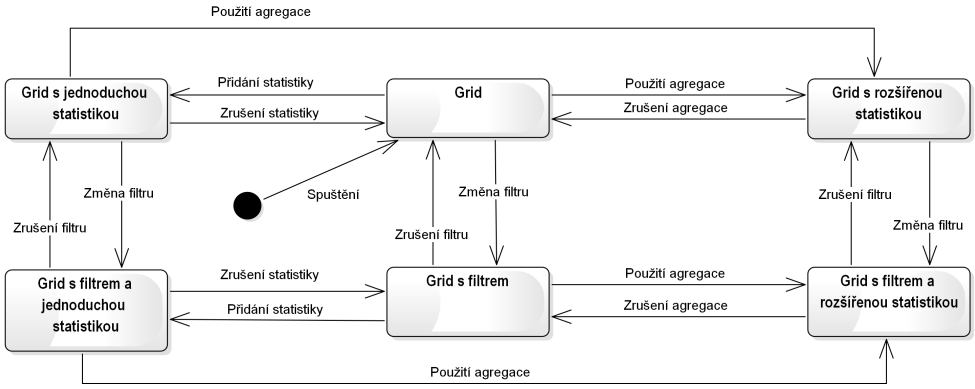
6.4 ClientApp

Před popisem samotné klientské aplikace ještě zmíníme další tenkou vrstvu, která slouží pro komunikaci klienta se službou spuštěnou na serveru.

[15] `https://msdn.microsoft.com/cs-cz/library/system.reflection.emit(v=vs.110).aspx` [citováno 11. května 2016]

[16] `https://msdn.microsoft.com/en-us/library/system.windows.controls.usercontrol(v=vs.110).aspx` [citováno 11. května 2016]

[17] `http://epplus.codeplex.com/` [citováno 11. května 2016]



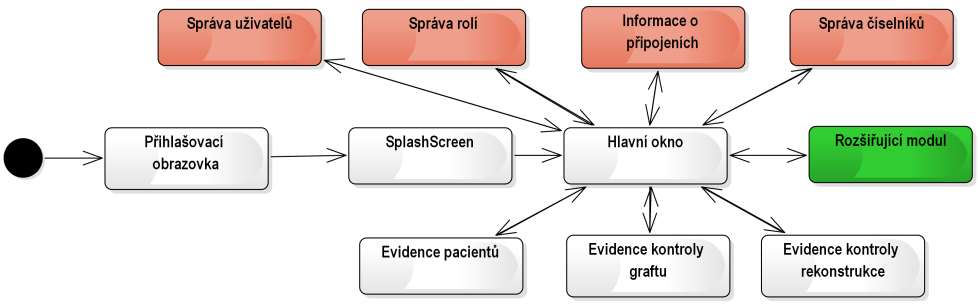
Obrázek 6.6: Diagram stavů EDG.

6.4.1 ClientProxy

Tato vrstva se jmenuje **ClientProxy**. Obsahuje třídu implementující stejné rozhraní, jako služba spuštěná na serveru. Klient pak může použít tuto proxy-implementaci rozhraní služby a pracovat tak se vzdálenou službou jako se standardními strukturami v paměti. Navíc Microsoft Visual Studio[18] umožňuje na základě dostupné WCF služby takové proxy třídy jednoduše vygenerovat.

6.4.2 Struktura aplikace

Struktura klientské aplikace je zobrazena na následujícím diagramu:



Obrázek 6.7: Diagram hlavních oken aplikace a možných přechodů mezi nimi. Červenou barvou jsou znázorněna okna přístupná pouze pro správce aplikace. Okno znázorněno zelenou barvou závisí na konkrétní implementaci rozšiřujícího modulu.

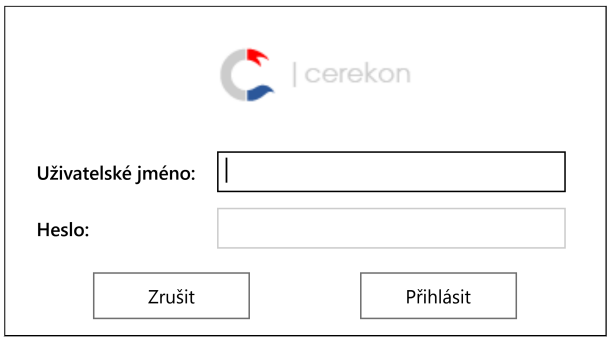
6.4.3 Přihlašovací obrazovka

Na přihlašovací obrazovce uživatel pouze vyplní své přihlašovací jméno a heslo. Stiskem tlačítka nebo klávesy enter se zadání potvrdí.

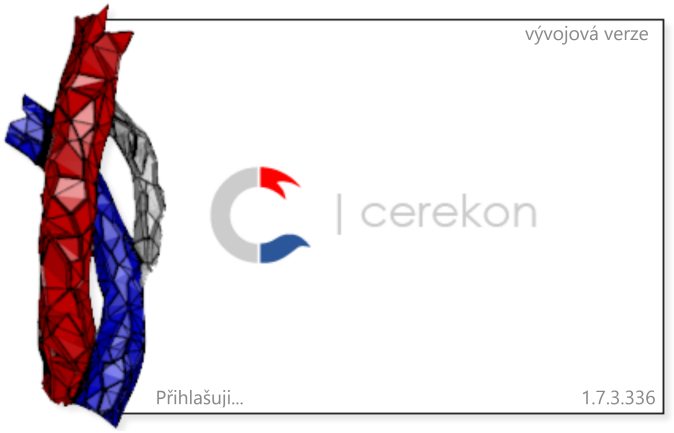
6.4.4 Splash screen

Splash screen je obrazovka, která se zobrazí typicky po spuštění nebo přihlášení do aplikace. Nejčastěji slouží jako informativní výstup, zatímco aplikace načítá

[18] `https://www.visualstudio.com/` [citováno 11. května 2016]



Obrázek 6.8: Přihlašovací obrazovka.



Obrázek 6.9: Splash screen.

data, knihovny nebo provádí jiné potřebné operace. Zobrazení splash screenu je jistě pro uživatele příjemnější, než načítání dat při spuštění. V takovém případě může být relativně velká časová prodleva mezi spuštěním aplikace (spuštěním procesu) a zobrazením UI. Uživatel se pak může domnívat, že aplikace neběží a spustí další instanci.

V našem systému je splash screen zobrazen během přihlašování uživatele, inicializaci jeho session, načítání některých dat a načítání rozšiřujících modulů. Upozorníme pouze na text zobrazující režim, ve kterém běží aplikační server. Tato hodnota se z AS načítá pomocí webového rozhraní. Je třeba si uvědomit, že všechna rozhraní standardní služby jsou přístupná až po inicializaci session. Informace o režimu je však žádoucí zobrazit i bez autentifikace (např. v situaci, kdy uživatel má přístup pouze k testovacímu prostředí a pokoušel by se přihlásit na produkční).

6.4.5 Hlavní okno

Po provedení všech potřebných operací se uživateli zobrazí hlavní okno. Při analýze uživatelského rozhraní jsme se rozhodli pro ovládání pomocí tzv. „pásu karet“, známějšího spíše pod anglickým názvem „ribbon“[19]. Naším cílem bylo implementovat rozhraní dostatečně intuitivní pro běžného uživatele. Zjistili jsme, že

[19] `https://msdn.microsoft.com/en-us/library/windows/desktop/dn742393(v=vs.85).aspx` [citováno 11. května 2016]

se ve VFN často používá balík Microsoft Office, pro který je rozhraní s ribbonem typické. Konzistentní způsob ovládání tak zlepšuje uživatelskou přívětivost.

Ačkoli WPF nabízí podporu pro vytváření aplikací s ribbonem[20], rozhodli jsme se pro použití knihovny **Fluent.Ribbon**[21]. Tato knihovna nabízí širší možnosti nastavení i lepší funkcionalitu.

V horní části hlavního okna se nachází ribbon s kartami pro jednotlivé funkční celky. Největší část okna tvoří záložky, z nichž na každé z nich je jeden EnhancedDataGrid zobrazující příslušný dataset. Uveďme seznam záložek:

přehled seznam všech vytvořených nebo upravených pacientů a kontrol

pacienti seznam pacientů

kontroly graftu seznam kontrol arteriovenózních graftů

kontroly rekonstrukce seznam kontrol cévních rekonstrukcí

Po dvojkliku na řádek se zobrazí postranní panel s detailními informacemi o objektu, který je řádkem reprezentován. Nebudeme na tomto místě detailně popisovat veškerou dostupnou funkcionalitu, to přenecháme uživatelské příručce (viz přílohy). Na příkladu jednoho okna pouze popíšeme postupy použité i na dalších místech aplikace.

6.4.6 Evidence kontroly graftu

Okno se skládá ze třech vizuálních komponent: pro pacienty, AV grafty a pro kontroly AV graftů. Každá z těchto komponent má svůj vlastní viewmodel. Viewmodel pro celé okno pak sdružuje všechny podřízené viewmodely a zprostředkovává komunikaci mezi nimi. Např. když viewmodel komponenty pro pacienty načte model pacienta, vyvolá událost, kterou zachytí viewmodel okna a předá viewmodelu komponenty pro AV grafty seznam AV graftů načteného pacienta. Viewmodel okna dále obsahuje vlastnosti potřebné pro celé okno (např. obsluha tlačítek) společně s kontrolou validace podřízených viewmodelů.

Všechny vizuální komponenty používají tzv. validační pravidla[22]. Validační pravidla umožňují kontrolovat uživatelem zadané hodnoty. V případě, že hodnota nesplňuje požadovaná kritéria, vrací negativní výsledek spolu se zprávou. Validátory se specifikují jako součást data bindingu. WPF používá i implicitní validátory. Např. pokud je textové pole připojené na vlastnost číselného typu a uživatel zadá řetězec, který není možné převést na číslo, implicitní validátor vrátí chybu.

Vedle již zmíněných validátorů, které validují vstup na úrovni uživatelského rozhraní, nové verze WPF nabízejí možnost použít rozhraní `INotifyDataErrorInfo`[23]. Toto rozhraní umožňuje provádět validace dat na úrovni viewmodelů.

[20] `https://msdn.microsoft.com/en-us/library/ff799534(v=vs.110).aspx` [citováno 11. května 2016]

[21] `http://fluentribbon.github.io/` [citováno 11. května 2016]

[22] `https://msdn.microsoft.com/en-us/library/system.windows.controls.validationrule(v=vs.100).aspx` [citováno 11. května 2016]

[23] `https://msdn.microsoft.com/cs-cz/library/system.componentmodel.inotifydataerrorinfo(v=vs.110).aspx` [citováno 11. května 2016]

Oba přístupy mají svá pozitiva i negativa. Validace na úrovni viewmodelu je „čistší“ z pohledu MVVM. Cenou však může být ztráta typové kontroly (která je nahrazena právě validací).Vysvětleme na jednoduchém příkladu:

Příklad Mějme textové pole, které chceme navázat na vlastnost celočíselného typu. V tuto chvíli nemůžeme validaci na viewmodelu použít, protože nevalidní hodnota se z UI do viewmodelu vůbec nedostane (do číselné vlastnosti nelze uložit řetězec). V tomto případě by bylo řešením změnit datový typ vlastnosti viewmodelu na řetězec (již zmíněná ztráta typové kontroly). Nevalidní hodnota je uložena do viewmodelu a poté je zahlášena chyba validace.

V knihovně Common (6.2) obsažený předek viewmodelů také implementuje rozhraní `INotifyDataErrorInfo` a tedy je možné jednoduše využívat oba typy validací.

Dalším důležitým krokem je prezentace výsledku validace, hlavně toho negativního, zpět do UI. Standardně se ve WPF používají tzv. „adornery“[24]. Implicitně má adorner po neúspěšné validaci podobu červeného orámování doplněného o popisek obsahující text výsledku validace. Toto základní chování nám vyhovuje a proto jej v aplikaci využíváme.

6.4.7 Administrace a Vývoj

Další dvě karty v ribbonu jsou přístupné pouze uživatelům s příslušným oprávněním. Využili jsme opět vlastnosti data bindingu, kde jsme implementovali speciální konvertor, který kontroluje oprávnění přihlášeného uživatele. Tento konvertor je přiřazený k data bindingu, který je nastaven pro vlastnost viditelnosti karty. Konvertor může mít v rámci data bindingu specifikovaný parametr a právě tímto parametrem je v našem případě GUID požadovaného oprávnění.

Administrace obsahuje:

- správu uživatelů
- správu uživatelských rolí
- správu číselníků
- přehled aktuálních připojení

Pro snazší implementaci administrace a jiné obdobné funkcionality jsme připravili vizuální komponentu, kterou jsme nazvali „AdministrableGrid“. Jedná se o jednoduchý grid o jednom sloupci, který je rozšířen o tři tlačítka: nový záznam, smazat a uložit. Připravený generický viewmodel pak umožňuje jednoduché použití s různými třídami.

6.4.8 Pluginy

Na poslední kartě se nachází aktivační tlačítka pro všechny načtené rozšiřující moduly (pluginy). Více o rozšiřitelnosti systému pojednává následující kapitola.

[24] `https://msdn.microsoft.com/en-us/library/ms743737(v=vs.100).aspx` [citováno 11. května 2016]

7. Rozšíření

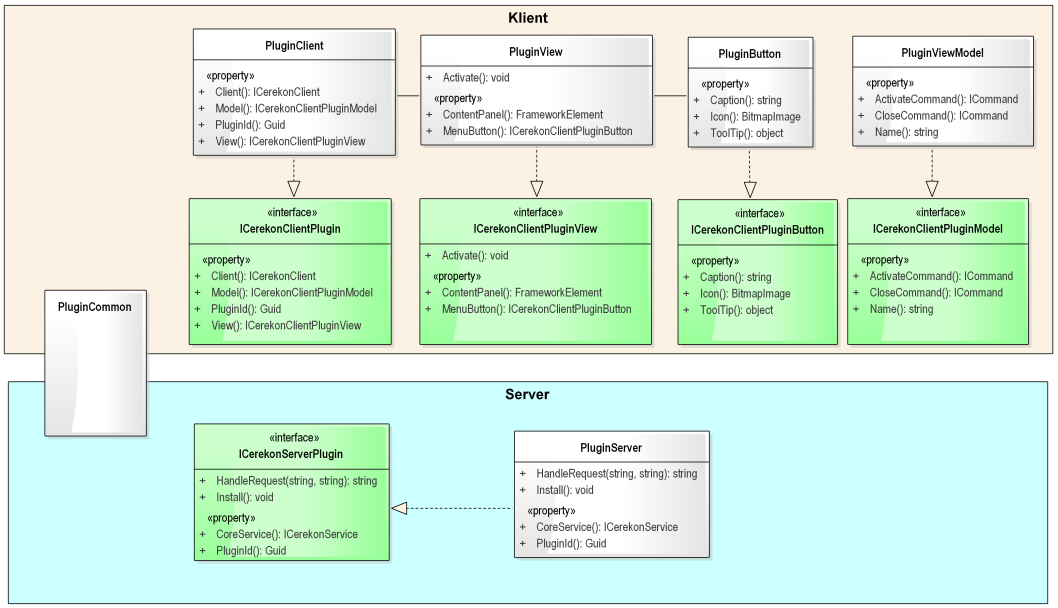
Jedním z našich cílů bylo připravit jednoduše rozšiřitelný systém. Tuto myšlenku na straně serveru podporuje také použití návrhových vzorů repozitář, business object a inversion of control, na straně klienta vzor MVVM a použití grafických komponent. Jedná se však o rozšiřitelnost na úrovni zdrojového kódu. Další rozšiřitelnost je možná i bez nutnosti kompilovat celý systém a to za pomocí **rozšiřujících modulů**, v aplikaci také nazývaných jako „pluginy“.

7.1 Architektura

Rozšiřující moduly mají podobu dynamicky linkovaných knihoven (DLL). Podle našeho návrhu uvažujeme, že jeden rozšiřující modul bude tvořen třemi knihovnami:

1. společné struktury
2. serverová část
3. klientská část

Základní strukturu rozšiřujícího modulu a rozhraní popisuje diagram 7.1:



Obrázek 7.1: Architektura rozšiřujícího modulu.

Každý rozšiřující modul je jednoznačně identifikován pomocí GUID (např. právě GUID modulu je typickým kandidátem na vyčlenění do sdílené části). Během inicializace session AS prohledá rekurzivně obsah složky `Plugins` a najde všechny třídy rozšiřujících modulů implementujících rozhraní `ICerekonServerPlugin`. Dále si vytvoří jejich instance a uloží si je indexované podle GUID. Služba spuštěná na AS zpřístupňuje metody `HandlePluginRequest` a `HandlePluginRequestQuery`, které jako parametry přijímají:

1. GUID modulu - slouží pro identifikaci volaného modulu
2. typ požadavku - slouží pro vnitřní identifikaci funkce modulu
3. parametry - další vnitřní parametry požadavku

Typ požadavku, parametry i návratová hodnota jsou typu `string`, kdy předpokládáme, že komunikace bude využívat serializaci založenou na řetězcích (např. XML nebo JSON).

Na straně klienta je princip fungování obdobný. Během spouštění aplikace prohledá složku `Plugins` a najde třídy implementující daná rozhraní. V případě klienta je potřeba vytvořit více objektů než na serveru. Konkrétně jde o objekty:

1. zastřešující celý modul
2. viewmodel
3. uživatelské rozhraní
   (a) tlačítko zobrazené v ribbonu
   (b) obsah záložky (hlavní UI)

V ribbonu je zvláštní záložka „Pluginy“, která obsahuje jedno tlačítko, tzv. aktivační tlačítko, pro každý načtený modul (ikonu a popis tlačítka poskytuje modul). Po stisknutí tlačítka se modul inicializuje a v hlavní části okna se zobrazí uživatelské rozhraní poskytnuté modulem.

Databáze

Serverová část rozšiřujícího modulu může k databázi přistupovat třemi způsoby:

1. přes standardní rozhraní služby
2. pomocí ORM
3. pomocí přímého přístupu

Z bezpečnostních důvodů však nechceme poskytovat modulům plný přístup k databázi. Modul by potenciálně mohl, třeba i neúmyslně, mazat nebo editovat záznamy databáze[1] nebo měnit její strukturu. Z těchto důvodů poskytujeme pro rozšiřující moduly separátní přístup k databázi. Přístup prostřednictvím služby zůstává beze změny. Ovšem použití ORM nebo přímého přístupu k databázi má omezená oprávnění. Takové připojení k databázi je izolované[2], autentifikované zvláštním databázovým uživatelem a je omezeno pouze na čtení dat. Omezení pouze na čtení dat může být, hlavně pro implementaci a efektivitu, velmi restriktivní. Chtěli jsme modulům poskytnout možnost definovat vlastní funkce nebo pohledy a tak předpokládáme v databázi vytvoření zvláštního schématu `plugin`. K tomuto schématu má modul neomezený přístup a může tak využít efektivní databázové konstrukce pro dotazování dat, ke kterým má přístup pouze pro čtení.

V samostatných kapitolách dále krátce popíšeme dva rozšiřující moduly, které jsme implementovali.

[1] Takové zásahy jsou možné i přes rozhraní služby, jedná se ale o systémový postup podléhající autentifikaci a autorizaci.

[2] Separátní databázová session.

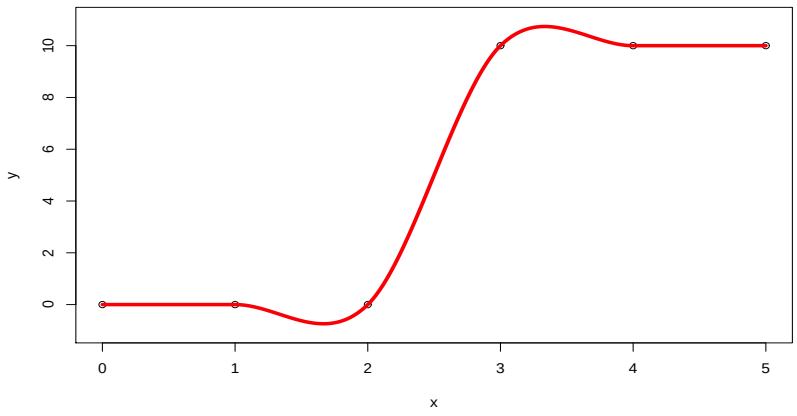
8. Modul Vizuální analýza AVG

Jedním z požadavků bylo zobrazit vývoj výsledků ultrazvukového vyšetření AVG pro jednotlivé pacienty. Hodnoty zjištěné Dopplerovskou ultrasonografií spolu s příslušnými jednotkami jsou:

- průměr přívodní artérie $[mm]$
- rychlost toku v přívodní artérii $[cm/s]$
- střední hodnota rychlosti toku v přívodní artérii $[cm/s]$
- průtok graftem $[ml/min]$
- reziduální průměr venózní anastomózy $[mm]$
- rychlost toku venózní anastomózou $[cm/s]$
- intima venózní anastomózy $[mm]$

Protože jde o číselné hodnoty měnící se v čase, rozhodli jsme se pro zobrazení v podobě spojnicového grafu. Pro implementaci grafů využíváme knihovnu **OxyPlot**[1]. Použití knihovny OxyPlot je poměrně přímočaré a kompatibilní s přístupem pomocí MVVM.

Jednou z užitečných vlastností OxyPlot v souvislosti se spojnicovými grafy je možnost interpolace. Interpolované křivky jsou pro uživatele přirozenější a přívětivější než prosté spojnice bodů[2]. OxyPlot nabízí kubickou interpolaci jako jedinou možnost volby. Nevýhodu kubické interpolace ilustruje obrázek 8.1.



Obrázek 8.1: Příklad kubické interpolace.

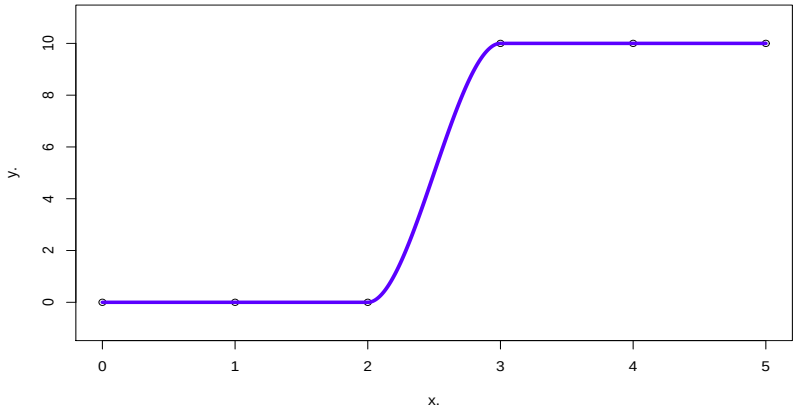
Jednoduše můžeme nahlédnout, že kubická interpolace nezachovává lokální extrémy. Zvláště pak situace mezi body se souřadnicemi [1,0] a [2,0] je z našeho

[1] `http://oxyplot.org/` [citováno 11. května 2016]

[2] Ačkoli i v tomto případě se jedná o interpolaci, a to konkrétně lineární.

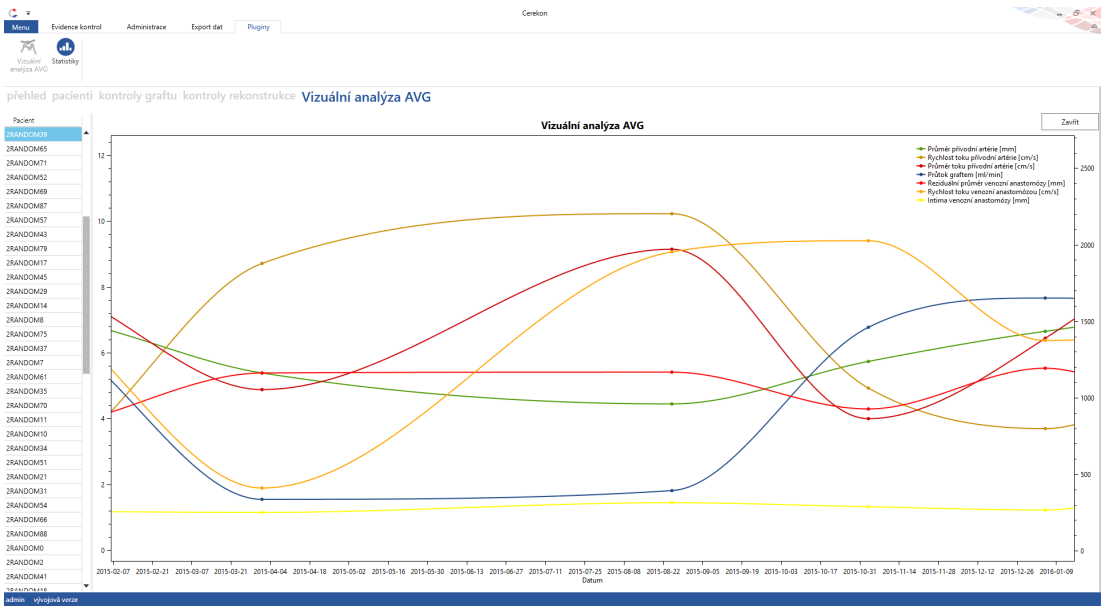
pohledu nežádoucí. Pokud bychom takto modelovali např. hyperplázii intimy, dostali bychom v určitém úseku záporné hodnoty síly intimy, což je nepřípustné.

Pro naše potřeby jsme tuto část knihovny OxyPlot upravili a místo kubické interpolace použili monotónní kubickou interpolaci[40]. Stejná data jako na předchozím obrázku s použitím monotónní kubické interpolace znázorňuje obrázek 8.2.



Obrázek 8.2: Příklad monotónní kubické interpolace.

Ovládání UI tohoto modulu je postaveno na vzoru master-detail[3]. Master záznamy jsou pacienti reprezentovaní svým identifikačním číslem zobrazení v gridu. Detail jsou data z kontrol AVG zobrazená v grafu. Domény výše zmíněných atributů se typicky budou řádově lišit. Zatímco síla intimy bude maximálně v jednotkách, hodnoty průtoků se mohou pohybovat v řádu tisíců. Z tohoto důvodu jsme do grafu umístili dvě osy Y, každou v jiném řádu.



Obrázek 8.3: Modul Vizuální analýza AVG.

[3] `https://msdn.microsoft.com/en-us/windows/uwp/controls-and-patterns/master-details` [citováno 11. května 2016]

9. Modul Statistiky

Druhý modul, který jsme implementovali, je tzv. modul statistik, nebo též přehled statistik. Budeme se zde věnovat hlavně implementaci, zatímco zdůvodnění, proč jsme zahrnuli právě takové statistiky, ponecháme poslední části této práce.

Z uživatelského pohledu je modul statistik rozdělen do tří částí:

1. základní
2. AV grafty
3. cévní rekonstrukce

9.1 Základní

Jako základní statistiky jsme připravili grafický přehled údajů podle zadaných požadavků. Ve stručnosti uveďme přehled zpracovávaných údajů:

- poměr pohlaví pacientů
- výskyt rizikových faktorů
- počty zákroků a kontrol
- histogramy dat z ultrazvukového vyšetření
- zastoupení lokalizace a typu graftu
- využití značek materiálů graftů
- zastoupení umístění operace
- korelační graf vybraných atributů

Z uživatelského i implementačního hlediska je zajímavá možnost měnit (omezovat) data, nad kterými se statistické údaje vyhodnocují. Dále toto budeme označovat jako **kontext** statistiky. Po analýze požadovaných statistických výstupů jsme se rozhodli jako kontext použít atributy evidované u pacientů. Nicméně změna množiny atributů tvořící kontext by v případě potřeby nebyla příliš složitá.

Implicitně se statistiky vyhodnocují v neomezeném kontextu, tedy nad všemi záznamy příslušné datové sady. Tato data může uživatel omezit za pomocí standardních filtrů, jako na dalších místech v aplikaci. Ovládání je tak konzistentní a intuitivní. Po aplikaci určitých kombinací filtrů mohou být některé statistické údaje nedostupné (v zadaném kontextu nejsou žádná data) nebo triviální (kontext omezený pouze na určitou hodnotu). Možnosti filtrování nikterak neomezujeme a necháváme jejich použití pouze na uvážení uživatele.

Z implementačního hlediska bylo vytvoření takovéto funkcionality relativně netriviální a proto ve zkratce popíšeme, jak jsme postupovali.

9.1.1 Kontext

Databázové dotazy potřebné k získání dat pro statistiku jsou často složitější než dotazy na dalších místech v aplikaci, tvořené jednoduše strukturou `SELECT ... FROM ... WHERE`. Dotazy pro statistiky jsou obvykle tvořeny vnořenými dotazy nebo agregacemi. Takové dotazy by bylo nesmírně složité generovat automaticky a i abstraktní mechanizmus pro definici takových dotazů na aplikační úrovni by byl značně netriviální. Z tohoto důvodu jsme v těchto případech odstoupili od odstínění aplikační vrstvy od databázové a dotazy jsou specifikovány v programovém kódu. Další komplikací je v tomto ohledu specifikace kontextu. Pokud chceme poskytnout možnost specifikovat filtry kontextu, musíme také přistoupit ke speciálnímu řešení. Standardní ORM, stejně jako náš framework Prosciutto, přidává selekci na konec dotazu, což je ve standardních případech dostačující. Pro analytické dotazy tento přístup nemusí být vždy použitelný, neboť atributy specifikované v kontextu nemusí být součástí výstupu. Do ORM jsme implementovali rozšíření, kde je možné specifikovat SQL výraz dotazu s vyznačeným místem, na které se má případná selekce aplikovat. Při dodržení potřebných jmenných konvencí (viz 4.1.1) je pak možné využít i automatické mapování výsledků na objekty.

Postupovali jsme tak, že explicitně definujeme dotazy pro jednotlivé statistiky. Kontext specifikujeme pomocí tzv. common table expression (CTE)[1], ve kterém definujeme místo, na které se doplní přeložená selekce. Tento CTE se dále používá pro omezení dat, které jsou výsledkem dotazu.

Kruhový diagram

V předchozí kapitole jsme popsali, jakým způsobem jsme rozšířili knihovnu OxyPlot pro použití spojnicových grafů. Pro vizualizaci dat reprezentující procentuální zastoupení jsme se rozhodli použít kruhový diagram[2]. Problémem v tomto případě bylo zobrazení legendy grafu. OxyPlot nabízel pouze legendu zobrazenou vždy u příslušné výseče grafu. Tento postup je přijatelný pouze do určité velikosti grafu v závislosti na počtu zobrazených výsečí. Pro naše potřeby jsme tento typ grafu v knihovně OxyPlot rozšířili o legendu zobrazenou vedle grafu. Pro každou výseč v grafu je v legendě zobrazen jeden řádek společně se čtvercem stejné barvy, jako je barva výseče v grafu.

Awaitable holder

Jako poslední zmíníme vizuální komponentu, kterou jsme nazvali „awaitable holder“. Při návrhu UI u modulu statistik jsme řešili situaci, kdy máme více statistik, které je potřeba vypočítat a je možné je počítat paralelně. Zároveň jsme chtěli na úrovni UI ošetřit stav, kdy výpočet statistiky právě probíhá. Protože jde o scénář, který se v modulu vyskytuje na více místech, rozhodli jsme se pro obecnější řešení. Tato komponenta nabízí následující funkcionalitu:

- je možné specifikovat, jestli je obsah načtený

[1] `https://technet.microsoft.com/en-us/library/ms190766(v=sql.105).aspx` [citováno 11. května 2016]

[2] Také známý pod označením „koláčový graf“.

- je možné nastavit tzv. *preloader*, který je zobrazen, pokud není obsah načtený
- obsah je možné uložit jako PNG obrázek do souboru

Použitím awaitable holderu na jednotlivé grafy získáme UI, které podporuje paralelní asynchronní zpracování a navíc je možné jednoduše grafy uložit do souboru.

9.2 AV grafy a cévní rekonstrukce

Zbylé dvě části modulu mají stejné rozhraní i funkčnost, liší se pouze zpracovávanými daty. I zde je možné pomocí filtru měnit kontext. Implementovali jsme dvě základní statistiky:

- chí-kvadrát test
- regresní model

Pro obě tyto statistiky jsme implementovali vizuální komponenty, aby bylo možné jejich znovupoužití (v našem případě dvakrát).

Chí-kvadrát statistika

V případě chí-kvadrát statistiky se UI skládá z výběru dvou atributů, pro které se bude statistika počítat, gridu, ve kterém se zobrazí kontingenční tabulka pro vybrané proměnné a konečně panelu, kde jsou zobrazeny hodnoty statistiky a výsledek testu hypotézy.

Regresní model

Regresní model se skládá z výběru cílové proměnné, pro kterou se vytvoří a ohodnotí lineární modely a zároveň se zobrazí histogram a základní statistické údaje cílové proměnné. Druhým krokem je výběr konkrétního modelu, který se dále vyhodnotí a získané údaje se zobrazí v grafech.

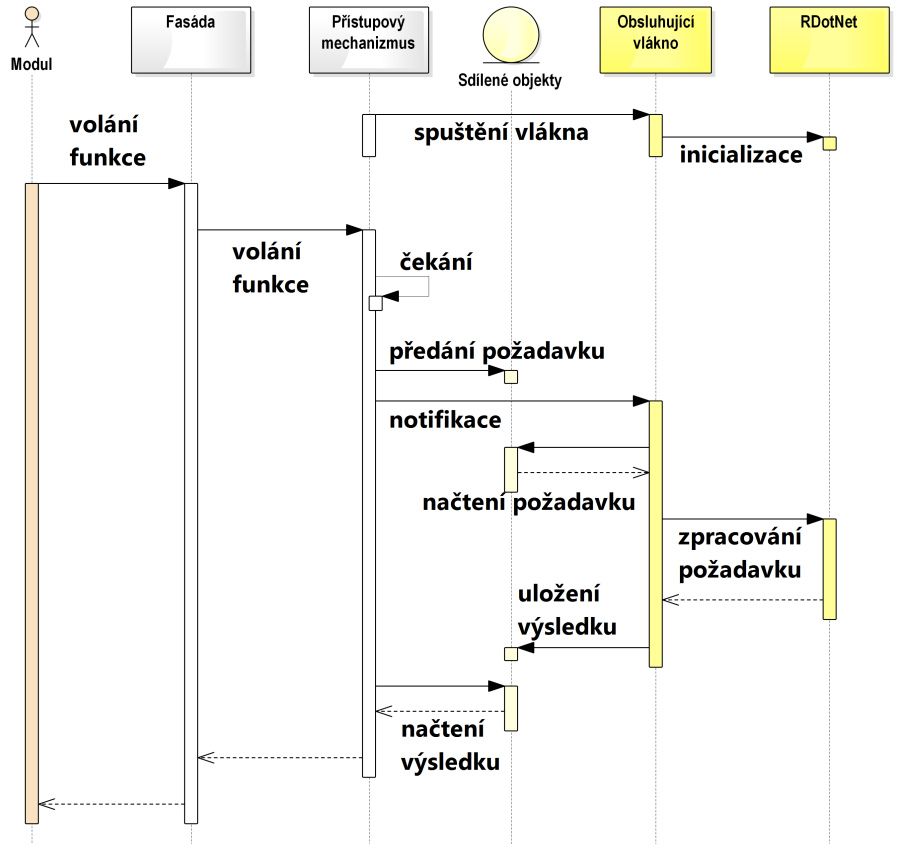
Integrace s R

Pro statistické výpočty jsme se rozhodli použít jazyk R[3], který se často používá jako alternativa k softwaru Matlab nebo SAS, protože je volně dostupný a nabízí mnoho funkčních balíčků. Statistické výpočty probíhají na aplikačním serveru, kde pro komunikaci s prostředím R používáme knihovnu RDotNet[4]. RDotNet poskytuje dva přístupy, které je ale typicky nezbytné pro efektivní práci kombinovat. První přístup plně integruje R do prostředí .NET. Datové typy R mají ekvivalentní třídy v .NETu, kde je s nimi možné dál pracovat. Druhou možností je vkládání příkazů přímo v jazyce R a jejich následné vyhodnocení.

[3] `https://www.r-project.org/` [citováno 11. května 2016]
[4] `https://rdotnet.codeplex.com/` [citováno 11. května 2016]

Knihovna RDotNet je koncipována jako tzv. *singleton*. To znamená, že po dobu trvání programu existuje nejvýše jedna instance. Navíc implementuje rozhraní `IDisposable`[5], které se používá, pokud objekt drží nějaké zdroje, například přístup k souboru nebo databázi. Právě vlastnosti singletonu nám způsobily další komplikace při použití. Docházelo totiž k situacím, kdy přístup k prostředí R končil chybou `AccessViolationException` způsobenou přístupem k poškozené paměti. Pozorováním se nám nepodařilo diagnostikovat konkrétní příčinu. Nicméně se domníváme, že se jedná o chybu způsobenou přístupem z různých vláken.

Nejjednodušším řešením tohoto problému by v jiných případech bylo uvolnění prostředí pomocí metody `Dispose` a jeho opětovným načtením. Takový postup jsme však nemohli použít, neboť je explicitně zakázán. Respektive pokus o druhou a každou další inicializaci prostředí R konči chybovou hláškou oznamující, že inicializace již jednou byla provedena. Tento postup zřejmě plyne z faktu, že knihovny je možné do aplikační domény v prostředí .NET nahrát, ale není možné jejich nahrání zrušit a nahrát je opětovně.



Obrázek 9.1: Sekvenční diagram synchronizovaného přístupu k prostředí R.

Využili jsme tedy jiný postup, kterým přístup z různých vláken eliminujeme. Pro prostředí R z knihovny RDotNet jsme definovali rozhraní, které na vyšší úrovni poskytuje potřebné statistické funkce. Toto rozhraní implementujeme a využíváme při tom volání prostředí R. Tato implementace pak běží ve speciálně

[5] `https://msdn.microsoft.com/cs-cz/library/system.idisposable(v=vs.110).aspx` [citováno 11. května 2016]

vytvořeném vlákně a přístup k ní je synchronizován semaforem. Požadavky se serializují ve frontě a čekají na zpracování dříve obdržených požadavků. Tento přístupový mechanizmus je skrytý za tzv. „fasádou“, což je třída implementující stejné rozhraní, která pouze předává požadavky přístupovému mechanizmu a vrací zpět výsledky. Průběh zpracování požadavku ilustruje sekvenční diagram na obrázku 9.1.

V některých případech jsme se rozhodli využít funkce jazyka R pro generování grafů místo vykreslování na klientovi pomocí OxyPlot. Jazyk R umožňuje jednoduše ukládat vykreslené grafy do PNG souborů. Požadovaný graf tedy uložíme do dočasného souboru, který dále načteme a obsah (obrázek) zakódujeme pomocí base64[6]. Jak jsme již dříve uvedli, zprávy rozšiřujících modulů mezi serverem a klientem jsou serializované jako řetězce, a proto je base64 vhodné kódování, přestože obecně není příliš efektivní.

Obrázek 9.2: Modul Statistiky.

[6] `https://tools.ietf.org/html/rfc4648` [citováno 11. května 2016]

Část III

Experimenty

10. Experimenty

V poslední kapitole naší práce se budeme zabývat experimenty, které je možné provádět nad zadanými daty. V první části objasníme důvod, proč jsme právě takto postupovali a v dalších částech popíšeme jednotlivé použité metody.

10.1 Motivace

Naším původním záměrem bylo se v této části práce věnovat analýze reálných dat, kterou bychom mohli získat nové cenné poznatky. Dobývání znalostí (data mining) (DZ), jehož prostředky jsme chtěli pro analýzu využít, se obvykle rozděluje na různé fáze v závislosti na použité metodice. Většina z nich zahrnuje fázi shromažďování a předzpracování dat do vhodné podoby. Uvádí se, že tato část je z celého procesu DZ potenciálně nejobtížnější a nejnákladnější (50 až 80 procent[41]). Toto pozorování se nám v praxi potvrdilo.

V našem případě je proces sběru a předzpracování dat implementovaný zadáváním dat pacientů, zákroků a kontrol do systému. Ačkoli jsme měli s poměrně značným předstihem přislíbeno od VFN (jakožto de facto zadavatele), že budeme mít k dispozici signifikantní množinu reálných dat, bohužel se tak ve skutečnosti nestalo. Podařilo se nám získat pouze desítky záznamů o kontrolách AV graftů a pouze jednotky o kontrolách rekonstrukcí. Pro ilustraci uveďme, že dle vyjádření VFN existují záznamy o cca 15000 zákrocích a ke každému z nich typicky i určitý počet záznamů o kontrolách. Příčinou nezadaných dat v systému jsou interní důvody VFN. Přes veškerou snahu i podporu ze strany II. chirurgické kliniky kardiovaskulární chirurgie, včetně jejího primáře, nám vedení VFN neumožnilo přístup k záznamům, které bychom mohli zadat do systému a posléze použít pro analýzu.

Místo analýzy reálných dat jsme se proto rozhodli pro implementaci takových statistických ukazatelů, které budou moci sloužit pro základní analýzu dat reálných. Při návrhu statistických ukazatelů jsme vycházeli buď z požadavků, které byly předem definovány, nebo jsme navrhli dostatečně obecnou metodu, kterou bude možné použít na datech, jejichž charakteristiky dosud neznáme. Výsledky použitých metod prezentujeme pomocí pseudonáhodných dat. Právě z tohoto důvodu jsme poslední kapitole dali název „Experimenty“.

Jak bylo zmíněno v předchozí kapitole, statistiky jsme implementovali do stejnojmenného rozšiřujícího modulu. Některé z nich nyní popíšeme obšírněji.

10.2 Základní

Na záložku „základní“ jsme umístili základní charakteristiky dat zadaných do systému. Jedná se o histogramy či procentuální vyjádření zastoupení hodnot kategoriálních atributů nebo jejich kombinací. Poslední prvek zobrazený na záložce je korelační graf hodnot vybraných atributů.

10.2.1 Korelační graf

Korelace vyjadřuje vztah mezi náhodnými veličinami. Jako kvantitativní míra jejich závislosti se nejčastěji používá **korelační koeficient**:

Definice 6 (Korelační koeficient)**.** *Buďte X, Y náhodné veličiny s konečnými druhými momenty a předpokládejme, že $var(X) > 0$ a $var(Y) > 0$. Pak (Pearsonův) korelační koeficient definujeme jako[42, str. 34]:*

$$\sigma = cor(X,Y) = \frac{cov(X,Y)}{\sqrt{var(X).var(Y)}}.$$

Korelační koeficient nabývá hodnot z intervalu $\langle -1,1 \rangle$. Hodnota:

0 znamená, že závislost proměnných nelze vyjádřit lineární funkcí nebo jsou nezávislé.

1 značí silnou korelaci (s rostoucí X lineárně roste Y).

-1 značí silnou antikorelaci (s rostoucí X lineárně klesá Y).

Korelační koeficient je tedy vhodným kandidátem na prvotní analýzu vztahu mezi dvěma atributy. Pokud budeme zkoumat míru korelace mezi všemi dvojicemi atributů, je vhodné použít **korelační matici**.

Definice 7 (Korelační matice)**.** *Korelační matice pro náhodné veličiny $X_1 ... X_n$ je čtvercová matice A velikosti $n \times n$, pro kterou platí:*

$$a_{i,j} = cor(X_i,X_j).$$

Z vlastností korelačního koeficientu plyne rovnost $cor(X,Y) = cor(Y,X)$, tedy korelační matice je zřejmě symetrická. Tohoto faktu je možné využít pro kompaktnější zobrazení matice.

Korelační graf je vizuální reprezentace korelační matice. Místo hodnot korelačního koeficientu se používá grafické znázornění, např. pomocí barvy nebo podílu vyplněné plochy prvku matice. Pro lepší analýzu korelačního grafu je možné zvolit takovou permutaci náhodných veličin $X_1 ... X_n$, aby obdobně korelované dvojice tvořily ve výsledné matici shluky.

Korelační matice počítáme v jazyce R funkcí `cor`. Pro vizualizaci výsledků, tedy vykreslení korelačního grafu, používáme funkci `corrplot` ze stejnojmenného balíčku[1].

V modulu Statistiky jsme korelační graf definovali pro atributy:

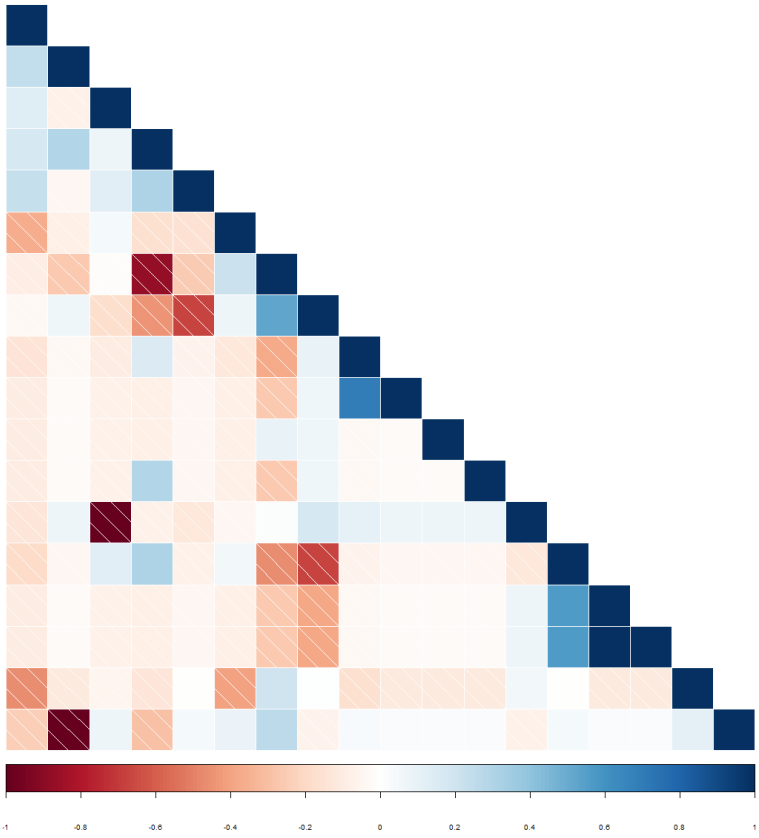
- primární onemocnění
- antikoagulační terapie
- antiagregační terapie
- materiál graftu

[1] `https://cran.r-project.org/web/packages/corrplot/index.html` [citováno 11. května 2016]

- příznak provedené endovaskulární intervence

Sledované atributy nejsou kvantitativní, ale kategoriální. Pro výpočet korelační matice jsme přistoupili k použití tzv. „falešných proměnných“[2][43]. Pro každou možnou hodnotu kategoriální proměnné (atributu) vytvoříme novou proměnnou, která bude nabývat hodnoty 1 právě tehdy, když bude mít původní kategoriální atribut hodnotu odpovídající dané falešné proměnné.

Formálněji zapsáno: Pro každý kategoriální atribut A, jehož aktuální doména $\{A_1, ..., A_N\}$ má velikost $N =\mid adom(X) \mid$, definujeme N proměnných $X_{A_1} ... X_{A_N}$, pro které platí: $X_{A_i} = 1 \Leftrightarrow A = A_i$, jinak $X_{A_i} = 0$.



Obrázek 10.1: Příklad korelačního grafu. Červená barva značí antikorelace, červená korelace. Hodnota korelačního koeficientu je reprezentovaná sytostí barvy.

Všechny atributy, které sledujeme v naší korelační matici, jsou kategoriální a nahrazujeme je množinou příslušných falešných proměnných. Výsledná matice je tedy řádu n, kde $n =\mid \bigcup_{A \in atributy} adom(A) \mid$. Vypočtené hodnoty nejsou korelace mezi atributy, ale mezi jednotlivými hodnotami atributů. Mohli bychom pozorovat kupříkladu závislosti mezi materiálem a antikoagulační terapií nebo antikoagulační terapií a prováděnými intervencemi. Právě sledování prováděných intervencí v závislosti na antikoagulační a antiagregační terapii (do konce odstavce již pouze terapie) by mělo být jedním z cenných analytických výstupů. Terapie je indikovaná pouze po určitou dobu po provedení zákroku. Vliv terapie na potřebu provádět intervence v delším časovém horizontu není znám. Pokud

[2]V anglicky psané literatuře známé pod názvem „dummy variable“.

by se ukázalo, že terapie je nezávislá na prováděných intervencích, umožnilo by to omezení terapie, která s sebou nese další rizika a vedlejší účinky.

10.3 AV grafty a cévní rekonstrukce

Stejně jako v případě implementace i na tomto místě zbylé dvě záložky modulu popíšeme dohromady. Použité postupy jsou shodné a liší se pouze vstupními daty.

10.3.1 Chí-kvadrát test

Jak již bylo řečeno, zjišťování závislostí mezi jednotlivými atributy je jedním ze základních analytických výstupů. Vedle analýzy korelací existují samozřejmě i jiné metody. Jednou z nich je χ^2 **test dobré shody**, který jsme implementovali. Na úvod shrneme některé základní definice.

χ^2 test je jedním z typu tzv. testování hypotéz. Testování hypotézy je ověřování tvrzení o pravděpodobnostním rozdělení (nebo jeho parametrech) pozorovaných dat. Formulované tvrzení je nulová hypotéza H_0, jejíž platnost budeme testem ověřovat. Pokud nulovou hypotézu zamítneme, znamená to připuštění platnosti alternativní hypotézy H_1. **Kritický obor** jsou hodnoty pozorování, pro které nulovou hypotézu zamítáme. Při testování nulové hypotézy se můžeme dopustit dvou typů chyb:

prvního druhu nulovou hypotézu zamítáme, přestože platí

druhého druhu nulovou hypotézu nezamítáme, přestože neplatí

V ideálním případě bychom chtěli výskyty obou typů chyb minimalizovat, což v praxi není možné. Chyba prvního druhu je obecně považována za závažnější a proto její míru zafixujeme a budeme tedy hledat test, jehož rozhodnutí budou chybná prvního druhu pouze v dané omezené míře. **Hladina významnosti** α testu je hodnota pravděpodobnosti chyby prvního druhu. Typicky volíme α malé, např. $0.05, 0.01$ nebo 0.001. **P-hodnota** určuje nejmenší hladinu významnosti, na které zamítneme nulovou hypotézu.

χ^2 test dobré shody je test o pravděpodobnostním rozdělení pozorované náhodné veličiny. χ^2 test porovnává četnosti výskytů pozorovaných hodnot s očekávanými četnostmi, které pochází z předpokládaného pravděpodobnostního rozdělení. Testem zjistíme, jestli jsou rozdíly mezi pozorovanými a předpokládanými četnostmi náhodné, nebo statisticky signifikantní. Pro jednoduchost budeme dále předpokládat, že všechny uvažované veličiny (atributy)[3] jsou kategoriální.

Testovat budeme vždy nezávislost dvou kategoriálních veličin, pro jejichž zobrazení použijeme tzv. **kontingenční tabulku**. Kontingenční tabulka zobrazuje četnosti výskytů jednotlivých hodnot sledovaných veličin.

Definice 8 (Kontingenční tabulka)**.** *Mějme atributy X a Y s aktuálními doménami*

$$adom(X) = \{X_1, X_2, ..., X_n\} \text{ a } adom(Y) = \{Y_1, Y_2, ..., Y_m\}.$$

[3] I v tomto kontextu můžeme pojmy veličiny a atributu považovat za ekvivalentní.

Pak kontingenční tabulka A pro veličiny X a Y obsahuje mn četností v n sloupcích a m řádcích. Pro prvek tabulky platí: $a_{i,j} = \#$ *pozorování s* $X = X_j$ *a* $Y = Y_i$*. Tabulka se také někdy rozšiřuje o* $(n+1)$*. sloupec a* $(m+1)$*. řádek. V tomto sloupci, resp. řádku, jsou sloupcové, resp. řádkové součty, jinak také nazývané jako „marginální hodnoty". Na pozici* $a_{m+1,n+1}$ *je pak celkový počet pozorování.*

DM/AH	Ano	Ne	Součet
Ano	13	0	**13**
Ne	26	2	**28**
Součet	**39**	**2**	**41**

Tabulka 10.1: Příklad kontingenční tabulky pro atributy „Arteriální hypertenze" a „Diabetes mellitus". V této podobě jde o speciální případ, tzv. čtyřpolní tabulku.

Při nezávislosti veličin X a Y bude očekávaná četnost $e_{i,j} = \frac{r_i . s_j}{n}$, kde r_i, resp. s_j je příslušný řádkový, resp. sloupcový součet a n je celkový počet pozorování. **Testová statistika** je funkce pozorovaných hodnot. Pomocí hodnoty testové statistiky budeme určovat výsledek testování hypotézy.

Definice 9 (χ^2*statistika*)**.** *Hodnotu* χ^2 *statistiky definujeme jako:*

$$\chi^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{(a_{i,j} - e_{i,j})^2}{e_{i,j}}.$$

Formulujeme nulovou a alternativní hypotézu:

H_0 : veličiny jsou nezávislé, tedy $P(X = X_i \wedge Y = Y_j) = P(X = X_i).P(Y = Y_j)$

H_1 : rozdíl mezi pozorovanými a očekávanými četnostmi není náhodný

Definice 10 (χ^2 test dobré shody)**.** *Nechť* X *a* Y *jsou kategoriální náhodné veličiny s aktuálními doménami velikosti* m*, resp.* n *a* $\forall_{i=1}^{m} \forall_{j=1}^{n} e_{i,j} \geq 5$*. Pak nulovou hypotézu nezávislosti* H_0 *zamítneme na hladině významnosti* α *tehdy a jen tehdy, když hodnota* χ^2 *statistiky pro příslušná pozorování je větší nebo rovna hodnotě* χ^2 *rozdělení s* $(m-1).(n-1)$ *stupni volnosti na hladině významnosti* α*.*

χ^2 test jsme implementovali pomocí funkce `CrossTable`[4] z balíku `gmodels`[5] v jazyce R. V implementaci zároveň předpokládáme, že pozorované hodnoty budou mít požadované četnosti. Test umožňujeme provést i pro atributy, které nejsou kategoriální. Takové atributy transformujeme na kategoriální tak, že jejich aktuální doménu rozdělíme na deset stejně velkých intervalů, které budou reprezentovat jednotlivé kategorie. Změna algoritmu rozdělování do kategorií, např. tak, aby v každé kategorii byl stejný počet záznamů, by nebyla složitá, stejně jako jiná rozšíření.

[4] V angličtině se pojem cross table používá pro kontingenční tabulku.

[5] `https://cran.r-project.org/web/packages/gmodels/` [citováno 11. května 2016]

10.3.2 Regresní model

Jako další statistický ukazatel jsme implementovali regresní model, resp. regresní analýzu. Cílem regresní analýzy je, obdobně jako v předchozím případě, zjištění vztahu mezi atributy, resp. vztahu atributu k jednomu nebo více atributů. Rozhodli jsme se implementovat model lineární regrese, který zjišťuje lineární závislost mezi proměnnými. V případě zjištěné závislosti jej lze použít i pro predikci nových výstupů na základě předložených vstupů. Zřejmě lineární model nemusí být optimální, mezi daty mohou být závislosti, které nelze aproximovat lineární funkcí a bylo by tedy lepší použít jiné techniky (např. polynomickou nebo logistickou regresi). Opět však připomeneme, že pracujeme nad daty, jejichž charakteristiky neznáme. Lineární regresní model považujeme za základní techniku a proto jsme se rozhodli jej implementovat.

Nejjednodušším případem je jednorozměrná lineární regrese.V tomto případě **vysvětlovanou proměnnou** predikujeme pomocí lineární funkce jedné **vysvětlující proměnné**, zapsáno jako:

$$\bar{y} = ax + b + \epsilon,$$

kde $\bar{y}$ je vysvětlovaná proměnná, x je vysvětlující proměnná, a,b jsou parametry modelu a konečně ϵ je náhodná složka vztahu mezi x a y.

Reziduum je rozdíl mezi skutečnou hodnotou vysvětlované proměnné a hodnotou predikovanou modelem, tedy pro i-tou položku datové sady $r_i = y_i - \bar{y_i}$. Lineární regrese se nejčastěji počítá pomocí **metody nejmenších čtverců**(MNC). Cílem MNC je najít parametry modelu takové, aby byl součet čtverců reziduí přes všechny položky z datové sady minimální. Tedy:

$$min \sum_{i=1}^{n} (y_i - \bar{y_i})^2 = min \sum_{i=1}^{n} (y_i - ax_i - b - \epsilon)^2.$$

Pomocí lineární regrese můžeme sledovat i závislost vysvětlované proměnné na více vysvětlujících proměnných. Jde tak o vícerozměrnou lineární regresi. Obdobně jako v jednorozměrném případě definujeme predikovanou hodnotu $\bar{y_i}$:

$$\bar{y_i} = a_0 + a_1 x_{i1} + a_2 x_{i2} + ... + a_k x_{ik} + \epsilon.$$

Pro regresní model je tedy důležitý výběr vysvětlujících proměnných. Možných modelů je zřejmě stejně, jako je počet podmnožin množiny proměnných. Ne všechny takové podmnožiny jsou však vhodné. Jedním z požadavků na dobrý výběr množiny je, aby jednotlivé proměnné byly nezávislé, resp. nebyly korelované. V opačném případě hovoříme o kolinearitě nebo multikolinearitě. Jeden z problémů, které způsobuje kolinearita, lze jednoduše vyjádřit. Jedná se o e maticový zápis vícerozměrné regrese. Pokud vektor predikovaných hodnot označíme jako $\vec{y} = (y_1, y_2, ..., y_n)$, vektor parametrů jako $\vec{a} = (a_0, a_1, ..., a_m)$ a vysvětlující proměnné jako matici

$$X' = \begin{pmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{pmatrix}, \text{ rozšířenou pro } a_0 \text{ jako } X = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,m} \end{pmatrix},$$

pak můžeme použít zápis $\vec{y}^T = X\vec{a}^T$. Pomocí MNC získáme parametry modelu jako[44]

$$\vec{a}^T = (X^T X)^{-1} X^T \vec{y}^T.$$

Pokud bude existovat silná kolinearita (korelace) mezi proměnnými, tedy jedna proměnná bude lineární kombinací jiných proměnných, pak matice X bude singulární a příslušná inverze k $X^T X$ nebude existovat. Také z pohledu výsledků predikce je vhodné mít nekorelované vysvětlující proměnné[6]. Obvyklým krokem při analýze dat je před vytvořením lineárního modelu provést výběr vhodných vysvětlujících proměnných. Korelované proměnné mohou tvořit tzv. komponenty. Jako vysvětlující proměnnou lze pak použít některou z komponenty nebo jejich kombinaci.

Postup v naší implementaci je do jisté míry obdobný. Nejdříve je vybrána cílová (vysvětlovaná) proměnná, která musí být kvantitativní. Následuje výběr prediktorů (vysvětlujících proměnných). Pro výběr vhodných prediktorů používáme funkci `regsubsets` z balíčku `leaps`[7]. Funkci předáme cílovou proměnnou a množinu potenciálních prediktorů spolu s daty. Výstupem jsou množiny prediktorů. Výstup je možné ovlivnit dalšími parametry: maximální počet prediktorů v jedné množině a počet množin se stejným počtem prediktorů. Modely jsou ohodnoceny pomocí Bayesovského informačního kritéria (BIC)[45]. Nastavení maximální velikosti modelu je důležité z uživatelského pohledu. Zřejmě výpočetní složitost roste exponenciálně s nastavenou maximální velikostí množiny prediktorů. Dále je třeba si uvědomit, že kategoriální proměnné se také v tomto případě transformují do falešných proměnných a množina potenciálních prediktorů navíc roste i s doménami kategoriálních proměnných. V této fázi naší implementace jsme omezili maximální velikost množiny prediktorů jako $max\{1, \lfloor 9 - log(|X|)\rfloor\}$, kde X je množina všech proměnných. Toto omezení se nám v praxi osvědčilo, kdy i pro velké množství proměnných (desítky) bylo možné spočítat výsledek v přijatelném čase[8].

Pro vybrané množiny prediktorů je pak možné vytvořit lineární regresní model. Jako výstup lineárního modelu slouží čtyři grafy:

graf reziduí
: zobrazuje závislost reziduí na predikovaných hodnotách. V ideálním případě není rozptyl reziduí velký a rezidua jsou náhodná.

rozdělení standardizovaných reziduí
: standardizované reziduum je podíl rezidua a směrodatné odchylky rezidua. Graf zobrazuje rozdělení standardizovaných reziduí proti normálnímu rozdělení. Hodnoty na diagonále značí standardní normální rozdělení.

[6] Na situaci se dá nahlédnout také tak, že korelované proměnné vysvětlují vztah stejným způsobem a jsou tudíž zbytečné.

[7] `https://cran.r-project.org/web/packages/leaps/index.html` [citováno 11. května 2016]

[8] Přijatelný čas je v tomto případě opět velmi relativní pojem. Pokud bychom hovořili např. o době výpočtu 5 minut, z pohledu datové analýzy nebo prediktivního modelování jde určitě o přijatelnou dobu. Z pohledu uživatele interaktivní aplikace, kterou implementujeme, je to však již neúnosně dlouhá doba.

graf standardizovaných reziduí
obdobně jako první graf zobrazuje závislost odmocniny standardizovaných reziduí na predikovaných hodnotách. Umožňuje ověřit homoskedasticitu reziduí.

graf vlivu vzdálených hodnot
na ose Y jsou opět hodnoty standardizovaných reziduí, na ose X je vliv hodnot na parametry regresního modelu. Může se stát, že některé hodnoty, které se značně odlišují od ostatních (tzv. vzdálené hodnoty), mají velký vliv na parametry modelu. Může se jednat o legitimní vlastnost, nebo naopak o šum v datech, který pak nepříznivě ovlivňuje výsledky predikce. Dalším ukazatelem v grafu je Cookova vzdálenost. Neformálně můžeme Cookovu vzdálenost definovat jako velikost změny parametrů modelu, pokud při jeho konstrukci vypustíme daný záznam. Tedy výše popsané vzdálené záznamy ovlivňující model budou mít velkou Cookovu vzdálenost a na grafu budou zobrazeny vpravo nahoře nebo dole.

Jako další údaje poskytujeme pro zvolenou vysvětlovanou proměnnou:

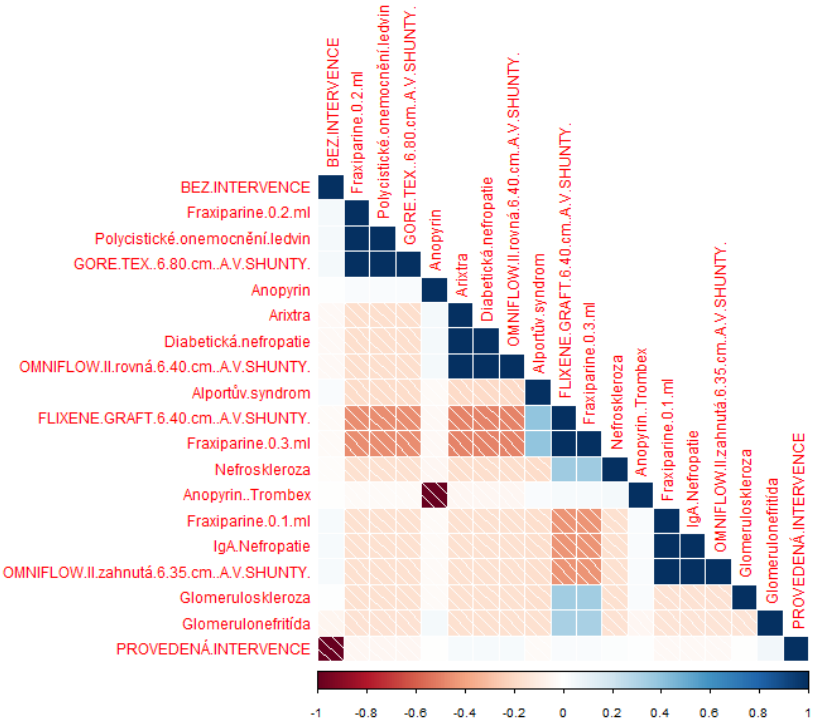
- histogram hodnot
- minimum
- maximum
- střední hodnotu
- medián

Implementovali jsme základní statistické ukazatele, které se typicky používají při analýze dat s neznámými charakteristikami, což je přesně situace, kterou jsme řešili. Poskytnuté údaje mohou sloužit jako podklady pro výběr dalšího postupu při prediktivním modelování.

10.4 Experimenty

Uvedené metody jsme testovali na pseudonáhodných datech, která jsme generovali podle rovnoměrného rozdělení. Zároveň jsme pro některé hodnoty vygenerovali korelovaná data (závislosti), abychom pak testovali, zda-li připravené vztahy představené metody odhalí. V následujících odstavcích popíšeme některé z výsledků těchto experimentů.

Korelační graf Na příkladu jsme ilustrovali korelační graf pro náhodná data (viz 10.1). Na obrázku 10.2 je korelační graf, kde jsme generovali antikoagulační terapii v závislosti na použitém materiálu graftu.



Obrázek 10.2: Příklad korelačního grafu. Můžeme pozorovat silnou korelaci mezi hodnotami antikoagulancií a materiálů graftu. Zároveň vidíme i očekávané antikorelace u zbylých hodnot. Nejsilnější antikorelace pozorujeme u antiagregancií a příznaku provedené intervence, které nabývají dvou možných hodnot.

Lineární model průtoku graftem Pro další příklad jsme vygenerovali data, kde je intima venózní anastomózy (VA) korelovaná, resp. antikorelovaná s průtokem graftem. Na seznamu vybraných modelů v tabulce 10.2 můžeme podle zahrnutých proměnných pozorovat, že proměnná „Intima VA“ je správně zahrnuta ve všech modelech (s výjimkou druhého a třetího, kde být nemůže). Navíc můžeme pozorovat zřejmé rozdíly v BIC. Charakteristiky modelu ukazuje obrázek 10.4.

Pokud použijeme χ^2-test dobré shody, dostaneme zřejmě jako výsledek zamítnutí nulové hypotézy nezávislosti a P-hodnota se bude velmi blížit nule. Příslušnou kontingenční tabulku zachycuje screenshot na obrázku 10.3.

Shluk odlehlých hodnot v lineárním modelu V dalším příkladu simulujeme situaci, kdy má určitá skupina vzorků silně specifickou vlastnost a tvoří tak shluk dat oddělený od ostatních záznamů. V literatuře nebo praxi se můžeme setkat s příkladem extrémně dobrých či špatných klientů nebo s daty obsahujícími chyby. V našem případě simulujeme specifickou sílu intimy u pacientů určitého věku (ale pro ilustraci bychom mohli zvolit i jiné atributy).

Protože ostatní hodnoty byly vygenerovány pseudonáhodně, lineární model proměnnou roku narození zahrnuje do všech přípustných modelů (čili detekuje závislost). Charakteristiky modelu můžeme vidět na obrázku 10.5.

Lineární model průtoku graftem podle průměru přívodní artérie (PA) a hypertenze Dále jsme vygenerovali závislost průtoku graftem na průměru

Stenóza rekonstrukce

Uzávěr rekonstrukce	<3	3-15	15-27	27-39	39-51	51-63	63-75	75-87	87-99	>=99
<11	22	25	4	0	0	0	0	0	0	0
11-22	0	21	13	0	0	0	0	0	0	0
22-33	0	1	17	18	1	0	0	0	0	0
33-44	0	0	2	22	17	0	0	0	0	0
44-55	0	0	0	0	30	19	0	0	0	0
55-66	0	0	0	0	1	26	11	0	0	0
66-77	0	0	0	0	0	3	20	21	0	0
77-88	0	0	0	0	0	0	7	26	14	0
>=88	0	0	0	0	0	0	0	4	26	4

Obrázek 10.3: Kontingenční tabulka pro uzávěr a stenózu rekonstrukce. Vygenerovaná závislost je zřejmá z rozložení dat podle diagonály. Hodnota χ^2-statistiky je přibližně 1295.44 při 72 stupních volnosti.

PA a arteriální hypertenzi, kdy:

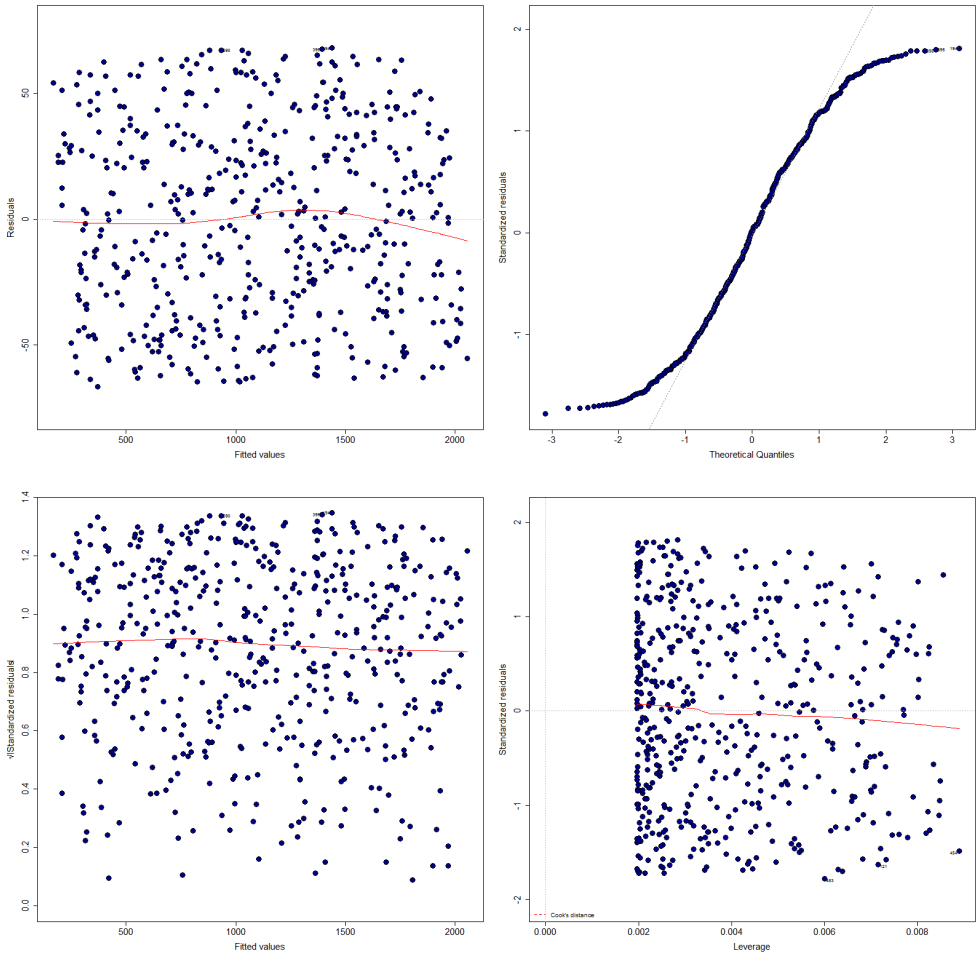
1. průtok $= 50 *$ průměr PA, pokud nemá AH
2. průtok $= 250 *$ průměr PA, pokud má AH

Taková závislost zřejmě není lineární, což můžeme jednoduše vidět na obrázku 10.6.

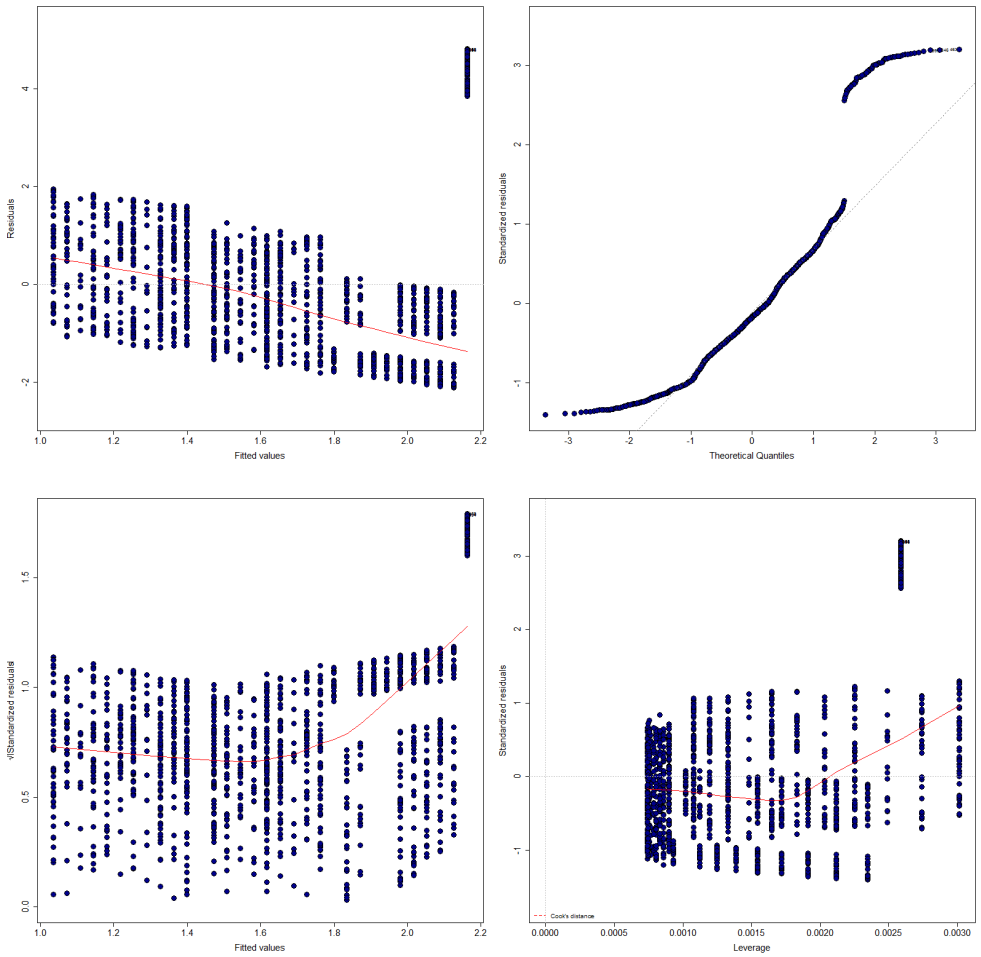
Přesto je lineární model schopen tuto závislost detekovat a přiřadit modelu s právě těmito proměnnými nejlepší skóre. Nelinearitu závislosti pak můžeme vysledovat z grafů modelu na obrázku 10.7.

model	BIC
Intima VA	-2647.31
Graft	6.58
PTA/Stent	7.91
Diabete mellitus, Intima VA	-2647.71
...	

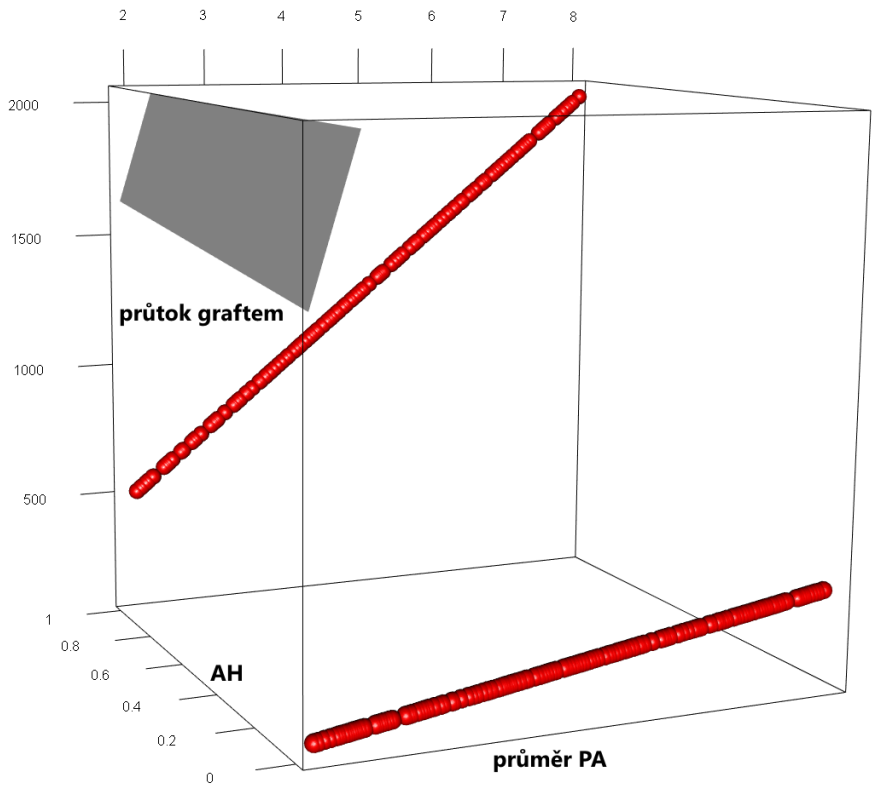
Tabulka 10.2: Lineární modely průtoku graftem. Všechny další modely zahrnovaly proměnnou „Intima VA“ a jejich skóre bylo obdobné.



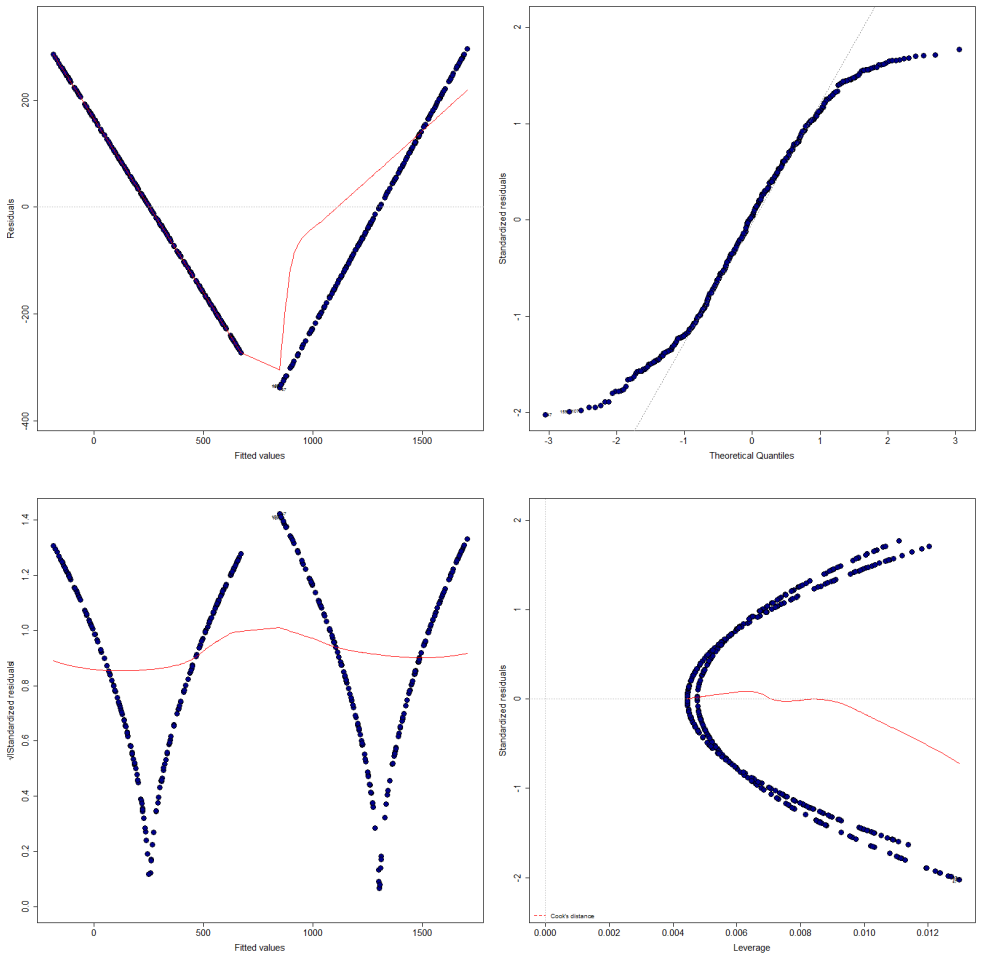
Obrázek 10.4: Lineární model průtoku graftem podle intimy venózní anastomózy. Můžeme pozorovat přesnou predikci na základě silné lineární závislosti. Pozorujeme malý rozptyl reziduí i nepřítomnout odlehlých hodnot. V grafu rozdělení reziduí pozorujeme v okrajových částech vychýlení způsobené použitím rovnoměrného rozdělení.



Obrázek 10.5: Shluk odlehlých hodnot v lineárním modelu. Ve všech grafech můžeme zřetelně pozorovat shluk odlišných vzorků. Lineární model zde není vhodný pro celou sadu dat, ale dává podnět k provedení shlukové analýzy (např. s využitím algoritmu k-means). Lineární model je pak možné aplikovat na získané shluky dat zvlášť.



Obrázek 10.6: Závislost průtoku graftem na hypertenzi a průměru PA. Šedou barvou je znázorněna regresní rovina.



Obrázek 10.7: Grafy lineárního modelu průtoku graftem v závislosti na arteriální hypertenzi a průměru PA. I ve výsledcích můžeme zřetelně pozorovat dva „shluky“(řady) dat.

11. Závěr

Na úvod naší práce jsme se zabývali popisem domény řešeného problému. Doménu tvoří cévní rekonstrukce a protetické hemodialyzační přístupy - arteriovenózní grafty. Náš popis nepředpokládá předchozí netriviální znalosti problematiky. Uvádíme však širší souvislosti umožňující lepší pochopení řešeného problému. Vytvořili jsme tak relativně stručné shrnutí, které není určené ani pro laiky bez zájmu o danou problematiku, ani pro odborníky. Tvoří tedy určitý kompromis, který jsme doposud v literatuře neobjevili.

Největší část práce jsme věnovali implementaci systému Cerekon. Systém jsme implementovali podle požadavků, které jsme získali z II. chirurgické kliniky kardiovaskulární chirurgie Všeobecné fakultní nemocnice v Praze, se kterou jsme spolupracovali. Výsledkem implementace je informační systém vrstevnaté architektury typu klient-server. V textu práce jsme se věnovali architektuře a netriviálním implementačním detailům systému. Systém jsme navrhli tak, aby byl jednoduše rozšiřitelný a to jak na úrovni implementace (pomocí různých návrhových vzorů), tak pomocí rozšíření bez nutnosti změn v systému samotném (pomocí rozšiřujících modulů).

Systém Cerekon jsme nasadili do produkčního prostředí na již zmíněné klinice VFN, kde byl k dispozici koncovým uživatelům. Za dobu produkčního nasazení systému jsme nezaznamenali závažnější komplikace, které by omezovaly jeho dostupnost klientům. V rámci navržené architektury bylo relativně jednoduché i nasazení aktualizací systému. Nejsložitější částí nasazení systému je instalace klientské aplikace na pracovní stanice koncových uživatelů, která musí probíhat plně v režii interního IT oddělení VFN.

Dříve zmíněnou rozšiřitelnost aplikace jsme demonstrovali na dvou modulech - vizuální analýze dat kontrol AV graftů a modulu statistik. Popsali jsme rovněž integraci s jazykem R, který poskytuje rozsáhlé možnosti pokročilého analytického zpracování.

Statistiky implementované v rámci modulu jsme popsali v poslední části práce. Protože jsme neměli k dispozici reálná data, přestože jejich sběr byl z naší strany umožněn produkčním nasazením systému, implementovali jsme experimentální statistické ukazatele. Jejich použití nad reálnými daty nemusí být optimální, jedná se však o základní údaje, které jsou užitečné při počáteční analýze dat. Konkrétně se jedná o analýzu korelací, test dobré shody a regresní analýzu. Ve všech případech jde o zjišťování případných vztahů mezi atributy. Právě zjištění nových vztahů mezi atributy evidovanými u pacientů nebo kontrol je do budoucna hlavní přidanou hodnotou systému. Pro zjištěné závislosti navíc bude možné provádět predikce dosud neznámých hodnot.

Dle našeho názoru se nám podařilo v rámci daného rozsahu všechny předsevzaté cíle splnit, s výjimkou analýzy reálných dat, jak již bylo zmíněno. Systém i definované procesy jsou však připraveny, takže po zaevidování dostatečného vzorku dat nebude složité analýzu provést.

12. Další práce

Systém do budoucna poskytuje možnost řady dalších rozšíření nebo optimalizací, kterým bychom se rádi věnovali. Z pohledu implementace to je další rozvoj systému jako softwarového díla podle dalších požadavků ze strany VFN. Bylo nám naznačeno, že by systém mohl být používán nejen pro analýzu historických dat, ale také pro on-line evidenci aktuálních záznamů.

Další možnosti pro vývoj poskytuje i ORM framework Prosciutto a vizuální komponenta EnhancedDataGrid. Obě tyto části systému by se mohly oddělit do samostatných open-source projektů.

A samozřejmě největším cílem do budoucna zůstává analýza reálných dat. Na základě reálných dat je možné do systému implementovat další on-line analytické nebo predikční nástroje.

Seznam použité literatury

[1] J. Norman Temple a P. Denis Burkitt. *Western diseases: their dietary prevention and reversibility*. Humana Press Inc., 1994.

[2] "Mayo Foundation for Medical Education and Research". Arteriosclerosis / atherosclerosis overview. Online: `http://www.mayoclinic.org/diseases-conditions/arteriosclerosis-atherosclerosis/home/ovc-20167019` [citováno 11. května 2016], 1 2016.

[3] "World health organization". Classification of atherosclerotic lesions. World health organization technical report series 143, World health organization, Palais des nations, Geneva, 1958.

[4] Richard Češka. *Interna*. Triton, Praha, 2010.

[5] Peter Libby, M. Paul Ridker, a Attilio Maseri. Clinical cardiology: New frontiers: Inflammation and atherosclerosis. *Circulation*, 105:1135–1143, 2002.

[6] S. Čech, M. Sedláčková, L. Ilkovics, D. Horký, I. Lauschová, a L. Krejčířová. Medatlas učební text a atlas. Online: `http://www.med.muni.cz/histology/MedAtlas_2/medatlas.html` [citováno 11. května 2016], 2006. Verze 2.1.

[7] Peter Libby, Paul M. Ridker, a Göran K. Hansson. Progress and challenges in translating the biology of atherosclerosis. *Nature*, 473(7347):317 – 325, 2011.

[8] Pavel Klener. *Vnitřní lékařství*. Galén, 2012.

[9] Jaroslav Masopust. Patogeneze aterosklerózy. Online: `http://dotdiag.cz/img/prednasky/atero.pdf` [citováno 11. května 2016].

[10] John A. Ambrose, FACC, a Rajat S. Barua. The pathophysiology of cigarette smoking and cardiovascular disease. *Journal of the American College of Cardiology*, 43(10), 2004.

[11] Jeffrey J. Siracuse a Elliot L. Chaikof. The pathogenesis of diabetic atherosclerosis. *Diagnosis an Management*, str. 13–26, 2012.

[12] M. Scott Grundy. Obesity, metabolic syndrome, and coronary atherosclerosis. *Circulation*, 105:2696–2698, 2002.

[13] Ali Al-Mamari. Atherosclerosis and physical activity. *Oman Medical Journal*, 24(3):173–178, 2009.

[14] Wayne R. Alexander. Hypertension and the pathogenesis of atherosclerosis: Oxidative stress and the mediation of arterial inflammatory response: A new perspective. *Hypertension*, 25(2):155–161, 1995.

[15] C.D. Liapis, editor. *Vascular Surgery*, chapter Development of Atherosclerosis for the Vascular Surgeon, str. 23–34. Springer Berlin Heidelberg NewYork, 2007.

[16] A. N. Assar a C. K. Zarins. Ruptured abdominal aortic aneurysm: a surgical emergency with many clinical presentations. *Postgraduate Medical Journal*, 85:268–273, 2009.

[17] Debora Karetová, Karel Roztočil, a Otto Herber. *Ischemická Choroba Dolních Končetin*. Společnost všeobecného lékařství ČLS JEP, 2011.

[18] Petr Bachleda. *Cévní chirurgie*. Univerzita Palackého v Olomouci, 2012.

[19] Jürgen Haase, Hans-Joachim Schäfers, Horst Sievert, a Ron Waksman. *Cardiovascular Interventions in Clinical Practice*. John Wiley & Sons, 2010.

[20] F. M. Ameli, M Stein, L. Aro, J. L. Provan, R. Gray, a H. Grosman. End-to-end versus end-to-side proximal anastomosis in aortobifemoral bypass surgery: does it matter? *Canadian journal of surgery*, 34(3):243–246, 1991.

[21] D. Mellière, J. Labastie, J. P. Becquemin, M. Kassab, a E. Paris. Proximal anastomosis in aortobifemoral bypass: end-to-end or end-to-side? *The Journal of Cardiovascular Surgery*, 31(1):77–80, 1990.

[22] Miroslav Zeman. *Speciální chirurgie*. Galén, 2006.

[23] Josef Veselka a Josef Rohn. *Kardiovaskulární medicína*. Facta Medica, 2015.

[24] "Mayo Foundation for Medical Education and Research". Chronic kidney disease. Online: `http://www.mayoclinic.org/diseases-conditions/kidney-disease/basics/causes/con-20026778` [citováno 11. května 2016].

[25] "National Kidney Foundation". About chronic kidney disease. Online: `https://www.kidney.org/kidneydisease/aboutckd` [citováno 11. května 2016].

[26] Atul Khasnis a Eamonn Molloy. Mimics of primary systemic vasculitis: Atherosclerosis & arteriolosclerosis. *International Journal of Clinical Rheumatology*, 4(5):1–13, 2009.

[27] Jan Vachek, Oskar Zakiyanov, a Vladimír Tesař. Chronické onemocnění ledvin. *Interní medicína pro praxi*, 14(3):107–110, 2012.

[28] K.-G. Fischer. Essentials of anticoagulation in hemodialysis. *Hemodialysis International*, 11:178–189, 2007.

[29] Milan Krajíček, Jan Peregrin, Miloslav Roček, a Pavel Šebesta. *Chirugická a intervenční léčba cévních onemocnění*. Grada Publishing a.s., 2007.

[30] Libor Janoušek a Peter Baláž. *Hemodialyzační arteriovenózní přístupy*. Grada Publishing a.s., 2011.

[31] Radojica Stolic. Most important chronic complications of arteriovenous fistulas for hemodialysis. *Medical Principles and Practice*, 22:220–228, 2013.

[32] Volker Mickley. Stenosis and thrombosis in haemodialysis fistulae and grafts: the surgeon's point of view. *Nephrology Dialysis Transplantation*, 19:309–311, 2004.

[33] Karl Wiegers a Joy Beatty. *Software Requirements, Third Edition*. Microsoft, 2013.

[34] Clustered and nonclustered indexes described. Online: `https://msdn.microsoft.com/en-us/library/ms190457.aspx` [citováno 11. května 2016].

[35] Nishith Pathak. *Pro WCF 4: Practical Microsoft SOA Implementation, Second Edition*. Apress, 2011.

[36] Martin Fowler. Inversion of control containers and the dependency injection pattern. Online: `http://www.martinfowler.com/articles/injection.html` [citováno 11. května 2016], 2004.

[37] Matthew D. Groves. *AOP in .NET, Practical Aspect-Oriented Programming*. MANNING Shelter Island, 2013.

[38] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, a John Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming (ECOOP), Finland*, 1997.

[39] Rudolf Pecinovský. *Java 7*. Grada Publishing a.s, 2012.

[40] Randall L. Gougherty, Alan Edelman, a James M. Hyman. Nonnegativity-, monotonicity-, or convexity-preserving cubic and quintic hermite interpolation. *Mathematics of Computation*, 52(186):471–494, 1989.

[41] Pipino Leo a David Kopcso. Data mining, dirty data and costs. *Proceedings of the Ninth International Conference on Information Quality*, 2004.

[42] Jiří Anděl. *Matematická statistika*. SNTL - Nakladatelství technické literatury, 1978.

[43] Max Kuhn a Kjell Johnson. *Applied Predictive Modelling*. Springer, 2013.

[44] Iveta Mrázová. Materiály k přednášce NDBI023.

[45] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistic*, 6(2):461–464, 1978.

Seznam obrázků

Seznam tabulek

Seznam zkratek

AA arteria axillaris - podpažní tepna. 18, 22

AAA aneurysma abdominální aorty. 13, 18, 22

AAP aneurysma popliteální artérie. 18

AF arteria fibularis - lýtková tepna. 18

AFC arteria femoralis comunnis - společná stehenní tepna. 18, 22

AFS arteria femoralis superficialis - povrchní stehenní tepna. 14, 18

AH arteriální hypertenze. 13, 24, 93

AOP aspektově orientované programování. 53, 54, 58

AP arteria poplitea - zákolenní tepna. 14, 18

API application programming interface. 64, 66

AS aplikační server. 49, 53, 55, 56, 58–60, 71, 74

ASL akutní selhání ledvin. 23

AT arteria tibialis - holenní tepna. 14

ATA arteria tibialis anterior - přední holenní tepna. 14, 18

ATP arteria tibialis posterior - zadní holenní tepna. 18

AV zkrat arteriovenózní zkrat. 24, 25

AVG arteriovenózní graft. 29, 34, 39, 76, 77

BIC Bayesovské informační kritérium. 90, 92, 94

CHRI chronická renální insuficience. 23

CHSL chronické selhání ledvin. 23, 24

CMP cévní mozková příhoda. 6

CTE common table expression. 79

DDL data definition language. 41

DLL dynamicky linkovaná knihovna. 74

DM diabetes mellitus. 12, 13, 24

DML data manipulation language. 41

DZ dobývání znalostí. 84

EDG EnhancedDataGrid. 64, 65, 69, 70

ePTFE expandovaný PTFE. 20, 25

FC pěnité buňky. 9

GF glomerulární filtrace. 24

GUID globally unique identifier - globálně unikátní identifikátor. 39, 40, 58, 73–75

HD hemodialýza. 24–26

ICHDK ischemická choroba dolních končetin. 14, 15, 18

ICHS ischemická choroba srdeční. 6

IoC Inversion of Control. 52, 53

JSON JavaScript object notation. 75

LDL nízkodenzitní lipoprotein. 9, 10

MDM aster data management. 37

MNC metoda nejmenších čtverců. 89, 90

MVVM Model-View-Viewmodel. 61–63, 65, 69, 73, 74, 76

NKF National Kidney Foundation. 23, 105

ORM objektově-relační mapování. 45–49, 55, 75, 79, 99, 110

PA přívodní artérie. 92, 93, 96, 97, 104

PET polyethylentereftalát. 20, 25

PTA Perkutánní transluminární angioplastika. 16, 17, 26, 103

RDBMS Relational Database Management System - relační databáze. 37

RF rizikový faktor . 11, 107

TK tlak krve. 22

UI uživatelské rozhraní (user interface). 61, 64, 69, 71, 73, 75, 77, 79, 80

VA venózní anastomóza. 92, 104

VC vena cephalica - cefalika. 19

VFN Všeobecná fakultní nemocnice v Praze. 5, 72, 84, 98, 99

VSM vena saphena magna - velká povrchová žíla. 19, 25

VSP vena saphena parva - malá povrchová žíla. 19

WCF Windows Communication Foundation. 49–51, 56, 57, 59, 70

WHO světová zdravotnická organizace. 6, 7

WPF Windows Presentation Foundation. 60–64, 72, 73

XML extensible markup language. 75

Část IV

Přílohy

A. Elektronické přílohy

Elektronické přílohy k této práci zahrnují zdrojové kódy, konfigurační a databázové skripty, návody a další soubory. V této kapitole popíšeme strukturu elektronických příloh.

readme.txt
: základní pokyny a informace o přílohách

Cerekon
: obsahuje zkompilované instalátory systému Cerekon

Source

Client
: obsahuje zdrojové kódy klientské aplikace

Common
: obsahuje zdrojové kódy modelu a ORM

CSharp_6_Examples
: obsahuje zdrojové kódy ukázek jazyka C# 6.0

Database
: obsahuje databázové skripty

Documentation
: obsahuje zdrojové kódy k programátorské dokumentaci

Installer
: obsahuje Inno Setup skripty pro vytváření instalátorů

Plugins
: obsahuje zdrojové kódy rozšiřujících modulů

Server
: obsahuje zdrojové kódy aplikačního serveru

Tex
: obsahuje LaTeXzdrojové kódy tohoto textu

Vendor
: obsahuje zdrojové kódy a knihovny třetích stran

Web
: obsahuje zdrojové kódy webových stránek

Text
: obsahuje text této práce v elektronické podobě a programátorskou dokumentaci

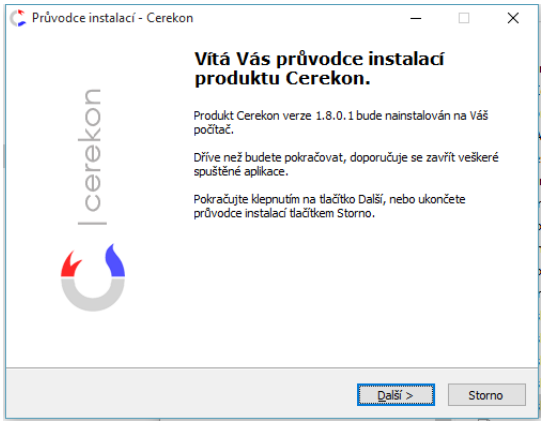
Vendor
: obsahuje instalátory potřebných komponent třetích stran

B. Uživatelská příručka

Následující kapitolu věnujeme uživatelské příručce klientské aplikace systému Cerekon. Popíšeme ovládání poskytované funkcionality skrze uživatelské rozhraní.

B.1 Instalace aplikace

Instalace aplikace je řešena pomocí instalátoru, jehož ovládání odpovídá standardním instalátorům v systému Windows. Úvodní obrazovku zachycuje obrázek B.1. Pro instalaci klientské aplikace nejsou vyžadována administrátorská oprávnění. Vyžadován je Microsoft .NET framework verze alespoň 4.5.

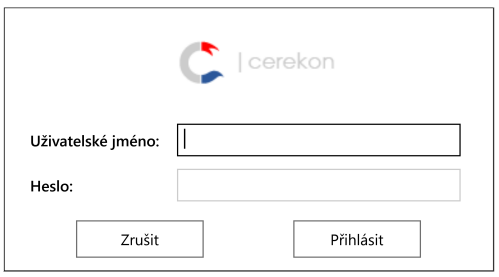


Obrázek B.1: Úvodní obrazovka instalátoru.

V dalších krocích instalátoru je možné zvolit umístění instalace[1] nebo vytvoření ikony na ploše.

B.2 Přihlášení

Po spuštění aplikace se zobrazí přihlašovací obrazovka (B.2). Po vyplnění uživatelského jména a hesla potvrdíte zadání stiskem tlačítka „Přihlásit“ nebo klávesou „Enter“.



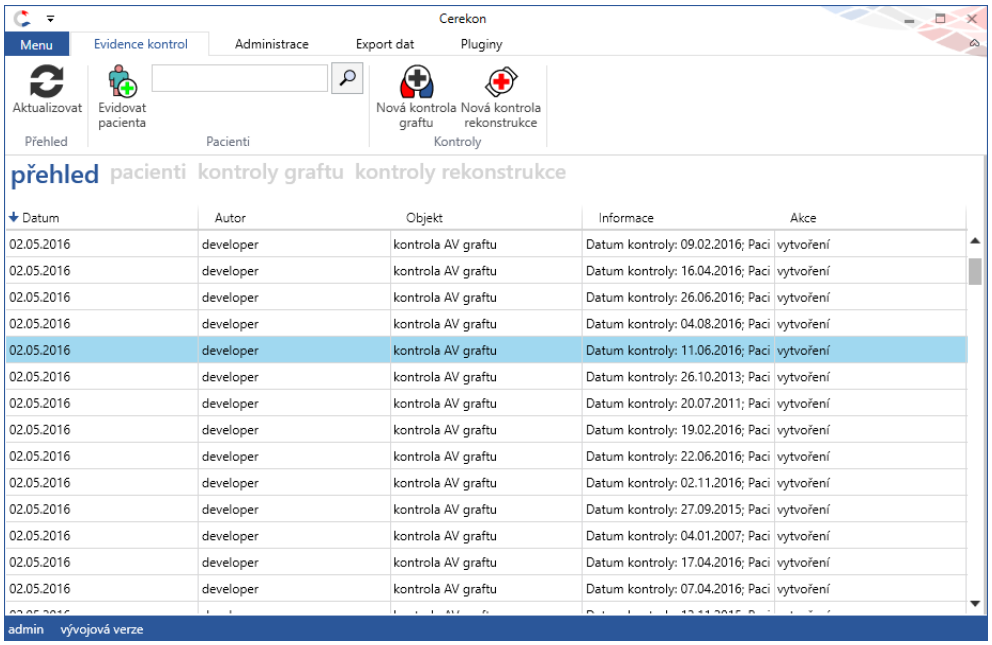
Obrázek B.2: Přihlašovací obrazovka.

V případě nesprávně zadaného hesla se zobrazí dialog, kde můžete své heslo nahradit nově vygenerovaným, které bude doručeno na Váš e-mail.

[1] Pro některá umístění mohou být vyžadována administrátorské oprávnění.

B.3 Hlavní okno

Po úspěšném přihlášení do aplikace se zobrazí hlavní okno (B.3). Hlavní okno se dělí na pás karet (ribbon), hlavní zobrazovací oblast a spodní panel zobrazující stavové informace.



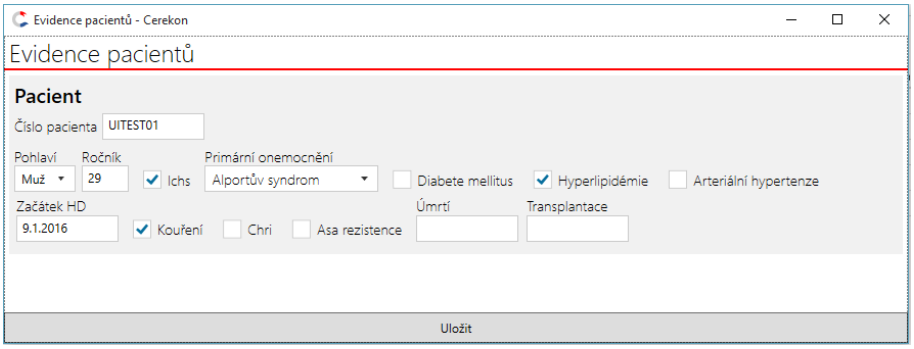
Obrázek B.3: Hlavní okno.

V hlavním okně je na pásu karet vybraná možnost „Evidence kontrol“, která obsahuje hlavní funkce aplikace.

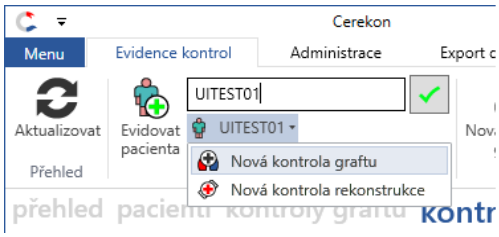
Aktualizovat
Obnoví grid provedených záznamů s názvem „přehled“.

Evidovat pacienta
Otevře dialog pro evidenci pacientů (B.4). Pacienty je možno evidovat nebo editovat nezávisle na provedených operacích nebo kontrolách.

Rychlá volba pacienta
Do textového pole rychlé volby pacienta (B.5) je možné zadat číslo pacienta a potvrzením pomocí tlačítka vpravo nebo stisknutím klávesy „Enter“ dojde k vyhledání pacienta s daným číslem. Pokud je takový pacient evidován, je možné ze zobrazené rozbalovací nabídky vybrat novou kontrolu rekonstrukce nebo graftu. Taková kontrola pak bude mít předvyplněného pacienta s dříve zadaným číslem.



Obrázek B.4: Evidence pacientů. Pacienta vyberete nebo vytvoříte zadáním jeho čísla do prvního textového pole a potvrzením stiskem klávesy „Enter“.



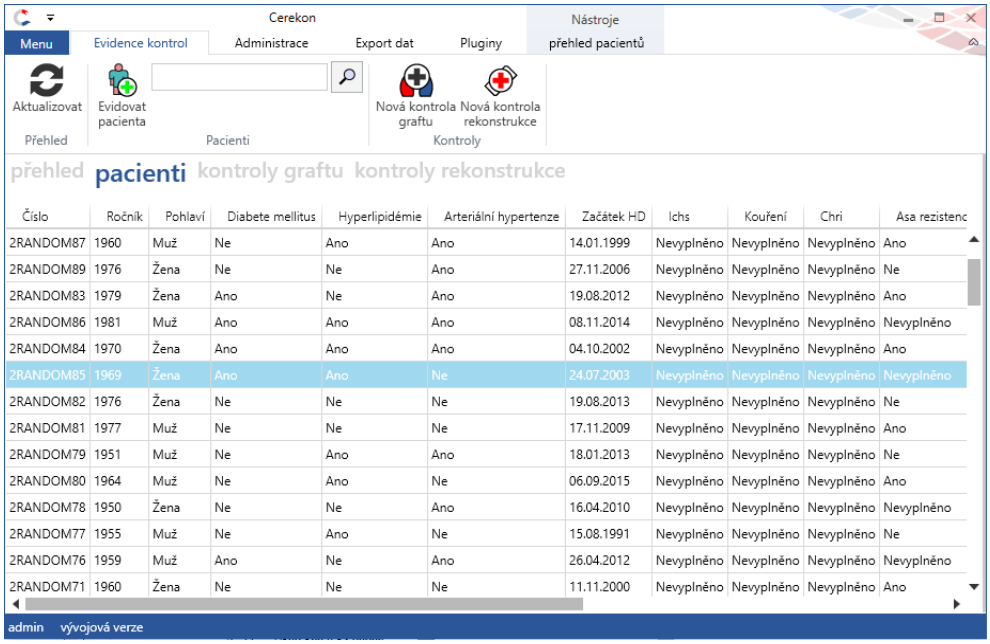
Obrázek B.5: Rychlá volba pacienta.

B.3.1 Přehledy

V hlavním okně aplikace se zobrazují přehledy:

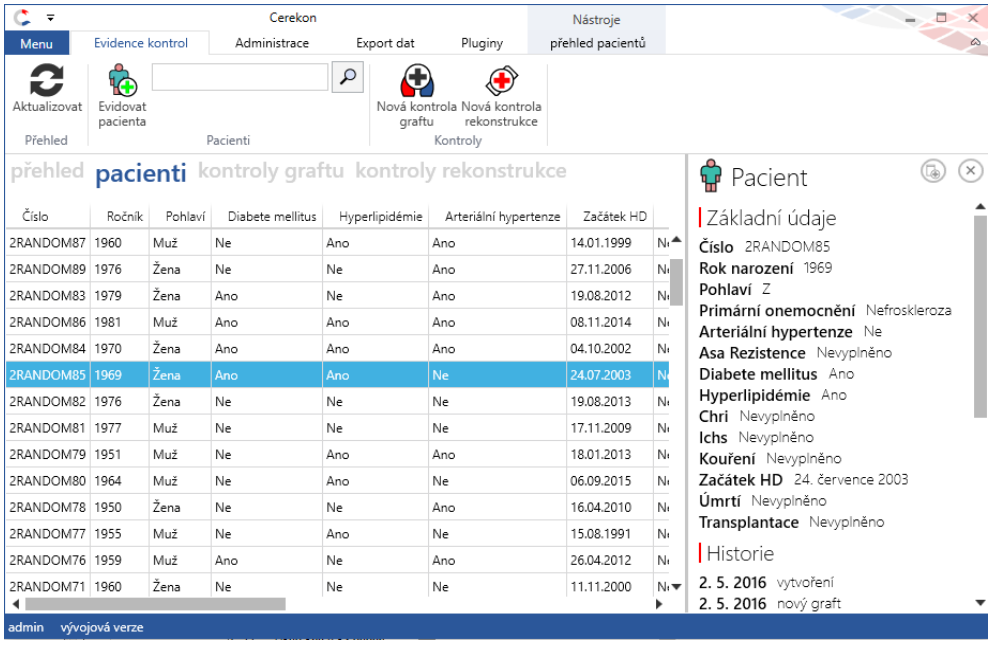
- **přehled** zobrazuje seznam zápisů (vytváření nových nebo úprava existujících záznamů) do evidence pro přihlášeného uživatele.
- **pacientů** (B.6)
- **kontrol AV graftů** (B.8)
- **kontrol rekonstrukcí**

Poklepáním na řádek v přehledu se zobrazí detail vybraného záznamu (B.7). Detail můžete zavřít tlačítkem „X“ vpravo nahoře, vedle kterého se nachází další tlačítko, které otevře detail záznamu v novém okně.



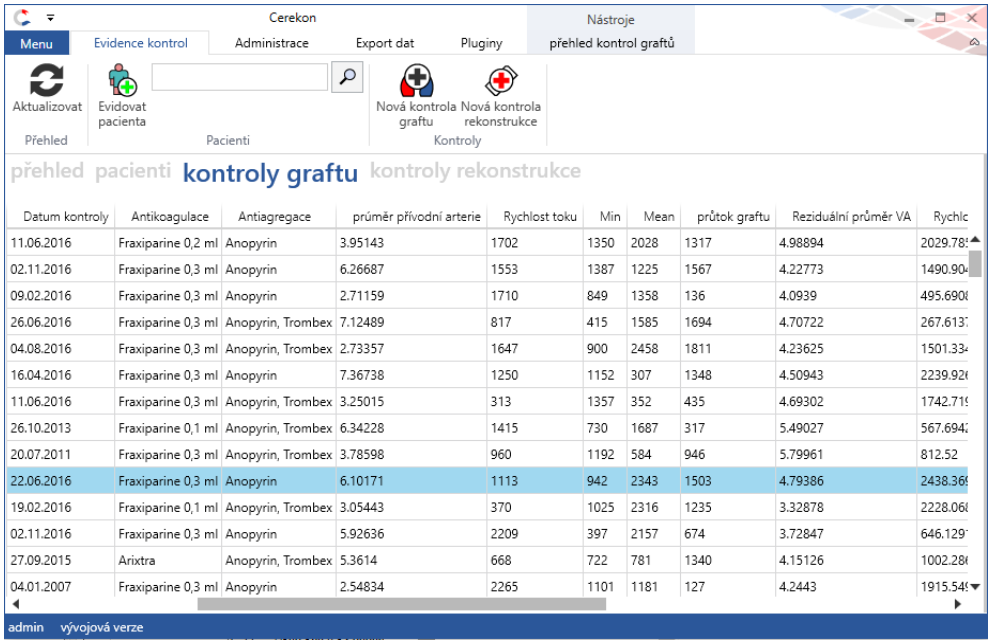
Obrázek B.6: Přehled pacientů.

Po zvolení příslušného přehledu, s výjimkou „přehledu“, se v pásu karet zobrazí nová položka s jediným tlačítkem „Aktualizovat“. Toto tlačítko slouží pro aktualizaci daného přehledu[2].



Obrázek B.7: Detail pacienta v přehledu.

[2]Tlačítko „Aktualizovat“ na záložce „Evidence kontrol“ obnoví záznamy právě v „Přehledu“.



Datum kontroly	Antikoagulace	Antiagregace	průměr přívodní arterie	Rychlost toku	Min	Mean	průtok graftu	Reziduální průměr VA	Rychlc
11.06.2016	Fraxiparine 0,2 ml	Anopyrin	3.95143	1702	1350	2028	1317	4.98894	2029.78
02.11.2016	Fraxiparine 0,3 ml	Anopyrin	6.26687	1553	1387	1225	1567	4.22773	1490.90
09.02.2016	Fraxiparine 0,3 ml	Anopyrin	2.71159	1710	849	1358	136	4.0939	495.690
26.06.2016	Fraxiparine 0,3 ml	Anopyrin, Trombex	7.12489	817	415	1585	1694	4.70722	267.613
04.08.2016	Fraxiparine 0,3 ml	Anopyrin, Trombex	2.73357	1647	900	2458	1811	4.23625	1501.33
16.04.2016	Fraxiparine 0,3 ml	Anopyrin	7.36738	1250	1152	307	1348	4.50943	2239.92
11.06.2016	Fraxiparine 0,3 ml	Anopyrin, Trombex	3.25015	313	1357	352	435	4.69302	1742.71
26.10.2013	Fraxiparine 0,1 ml	Anopyrin, Trombex	6.34228	1415	730	1687	317	5.49027	567.694
20.07.2011	Fraxiparine 0,3 ml	Anopyrin, Trombex	3.78598	960	1192	584	946	5.79961	812.52
22.06.2016	Fraxiparine 0,3 ml	Anopyrin	6.10171	1113	942	2343	1503	4.79386	2438.36
19.02.2016	Fraxiparine 0,1 ml	Anopyrin, Trombex	3.05443	370	1025	2316	1235	3.32878	2228.06
02.11.2016	Fraxiparine 0,3 ml	Anopyrin	5.92636	2209	397	2157	674	3.72847	646.129
27.09.2015	Arixtra	Anopyrin, Trombex	5.3614	668	722	781	1340	4.15126	1002.28
04.01.2007	Fraxiparine 0,3 ml	Anopyrin	2.54834	2265	1101	1181	127	4.2443	1915.54

Obrázek B.8: Přehled kontrol AV graftů.

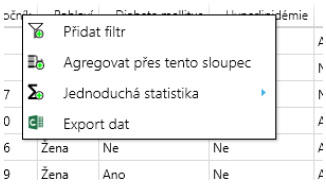
Záznamy v přehledech je možné filtrovat a definovat nad nimi jednoduché nebo rozšířené statistiky. Aktuální možnost pro každý sloupec se zobrazí v kontextové nabídce po kliknutí pravým tlačítkem kurzoru na záhlaví příslušného sloupce. Kontextová nabídka může mít podobu jako na obrázku B.9, tedy:

- přidání filtru
- výběr sloupce pro agregaci
- výběr/zrušení jednoduché statistiky
- export dat

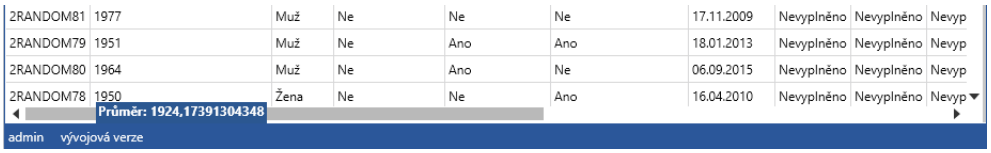
Pro různé typy sloupců jsou k dispozici různé jednoduché statistiky:

- počet
- minimum
- maximum
- součet
- aritmetický průměr
- zastoupení logických hodnot

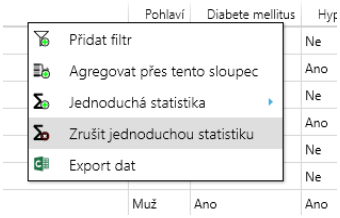
Po výběru jednoduché statistiky se výsledek zobrazí v patičce sloupce (B.10). Jednoduchou statistiku je možné zrušit opět prostřednictvím kontextové nabídky (B.11).



Obrázek B.9: Kontextová nabídka sloupce.

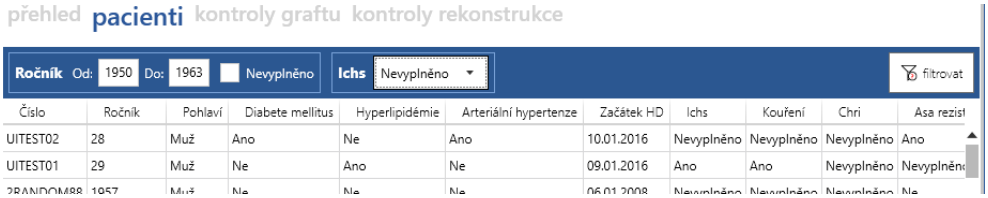


Obrázek B.10: Příklad výsledku jednoduché statistiky.



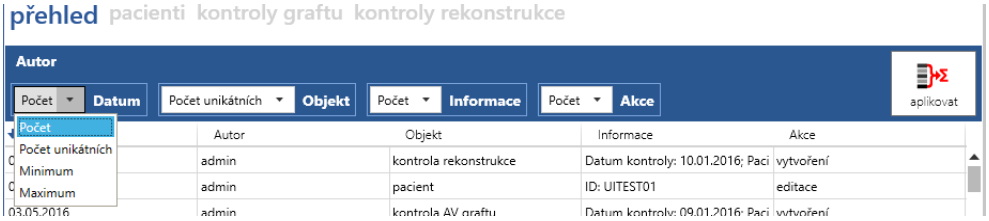
Obrázek B.11: Zrušení jednoduché statistiky v kontextové nabídce.

Rovněž filtry se přidávají pomocí kontextové nabídky, konkrétně pomocí volby „Přidat filtr“. Tato volba přidá zvolený sloupec, tedy ten, na jehož záhlaví se kontextová nabídka nachází, jako filtrovací sloupec. Filtr se aplikuje pomocí tlačítka vpravo nebo stiskem klávesy „Enter“. Filtr odstraníte kliknutím pravého tlačítka kurzoru na příslušný filtr.



Obrázek B.12: Příklad filtrů.

Stejným způsobem, tedy přes kontextovou nabídku, se definuje rozšířená statistika. Statistiku definujete pomocí seskupujících (agregačních) a agregovaných sloupců. Prvním krokem je zvolení prvního agregačního sloupce pomocí kontextové nabídky. Všechny zbylé sloupce se pak budou chovat jako agregované sloupce. Agregovaný sloupec je možno převést na agregační pomocí kontextové nabídky sloupce. Pro agregované sloupce je možné vybírat různé agregační operace. Nabídka agregačních operací je závislá na typu sloupce. Rozšířenou statistiku vyhodnotíte stiskem tlačítka na pravé straně panelu. Agregační sloupec odeberete, tedy změníte v agregovaný, stiskem pravého tlačítka kurzoru na název agregačního sloupce v panelu, kde se zobrazují agregační sloupce.



Obrázek B.13: Příklad rozšířené statistiky s jednín agregačním a čtyřmi agregovanými sloupci.

B.3.2 Menu

Tlačítkem vlevo nahoře je možné zobrazit aplikační menu. V rámci aplikačního menu je možné:

- zobrazit nápovědu
- přejít na webové stránky systému
- změnit své heslo

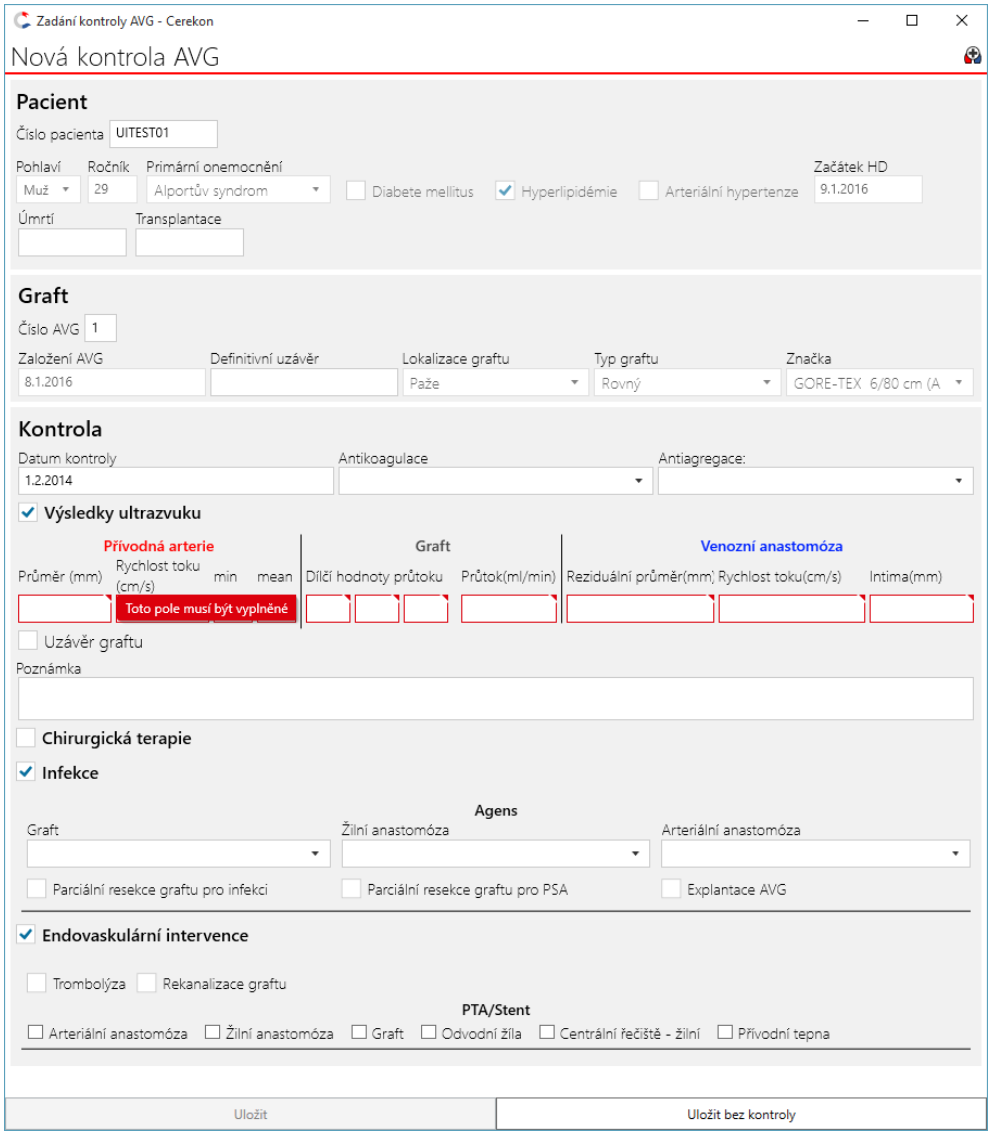
Heslo změníte zadáním stávajícího hesla, nového hesla, potvrzením nového hesla a stiskem tlačítka vpravo. O úspěšné změně hesla informuje zobrazený zelený nápis, o chybě červený nápis.

B.4 Evidence kontrol graftů

Stisknutím tlačítka „Nová kontrola graftu“ na pásu karet v hlavním okně se zobrazí dialog evidence nové kontroly AV graftu. Postupně vyplňte požadované údaje o pacientovi a graftu. Nyní máte dvě možnosti: uložit pacienta a graft nebo uložit pacienta, graft a kontrolu, kterou je potřeba nejdříve požadovaným způsobem zadat. Požadovaná pole jsou označena červenou barvou. Stejně tak je červenou barvou vyznačeno, pokud je vyplněná hodnota neplatná. V případě tučně zvýrazněných zaškrtávacích polí se po jejich označení zobrazí další možnosti zadání:

- výsledky ultrazvuku
- chirurgická terapie
- endovaskulární intervence
- infekce

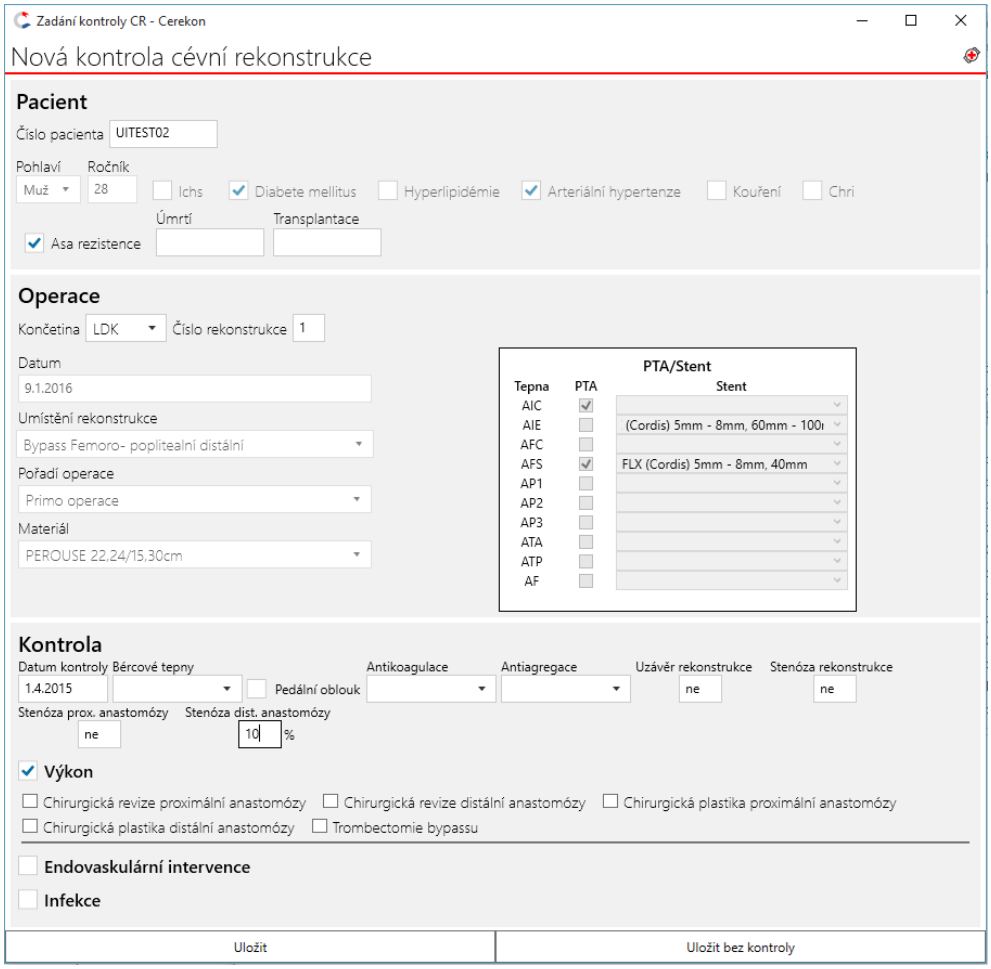
Po úspěšném uložení dat se zobrazí informace o uložení a dialog je zavřen.



Obrázek B.14: Dialog zadání kontroly AV graftu.

B.5 Evidence kontrol rekonstrukcí

Stisknutím tlačítka „Nová kontrola rekonstrukce“ na pásu karet v hlavním okně se zobrazí dialog evidence nové kontroly cévní rekonstrukce. Postup vyplnění je obdobný jako v případě kontroly AV graftu. Opět je zde možnost uložit pacienta a operaci bez kontroly nebo se zadanou kontrolou.

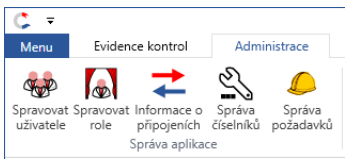


Obrázek B.15: Dialog zadání kontroly cévní rekonstrukce.

B.6 Administrace

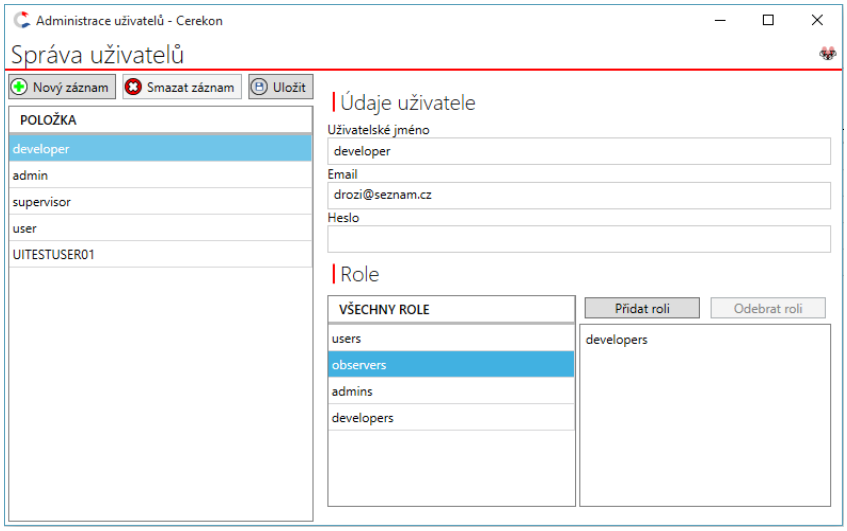
Pokud máte příslušná oprávnění, jednou z položek na pásu karet je i „Administrace“. Administrace, tedy správa aplikace, se skládá z:

- správa uživatelů
- správa rolí
- informace o připojeních
- správa číselníků
- správa požadavků



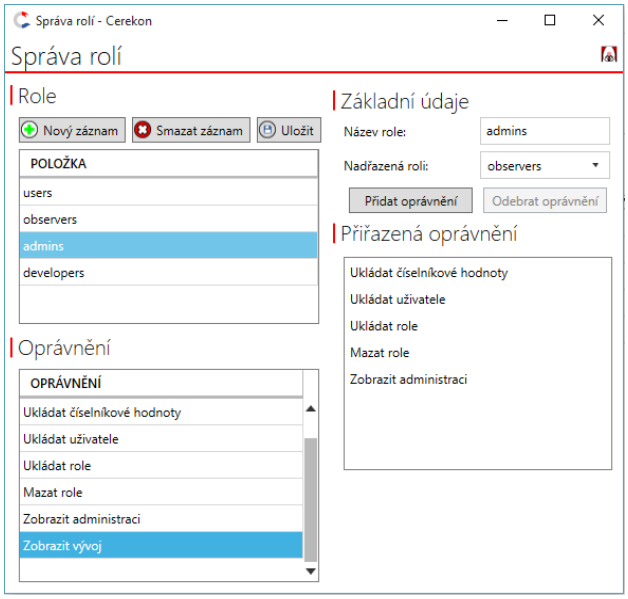
Obrázek B.16: Položky administrace systému.

Správa uživatelů (B.17) umožňuje přidávat a upravovat uživatele. Uživatelé jsou zobrazeni v tabulce vlevo. Tlačítky nad tabulkou je možno přidávat nové záznamy nebo ukládat provedené úpravy. Po výběru konkrétního uživatele se v pravé části okna zobrazí detailní údaje uživatele, které je možné editovat. Na tomto místě se také uživateli přiřazují nebo odebírají role.



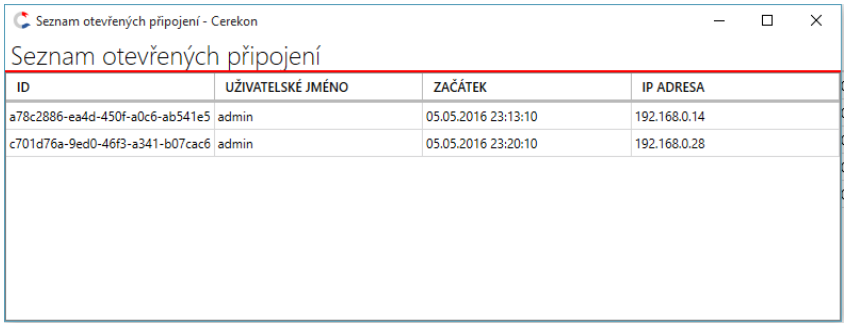
Obrázek B.17: Okno správy uživatelů.

Správa rolí (B.18) slouží pro vytváření a úpravy uživatelských rolí a nastavení oprávnění pro jednotlivé role. Ovládání záznamů rolí v levé horní části okna je obdobné jako v administraci uživatelů. V levé dolní části okna je seznam dostupných oprávnění. Ten není možné měnit, oprávnění jsou součástí implementace systému. Po výběru role se v pravé části zobrazí detaily dané role: název, nadřazenost roli a přiřazená oprávnění. Nadřazenost roli znamená, že daná role přebírá navíc všechna oprávnění role, které je nadřazená.



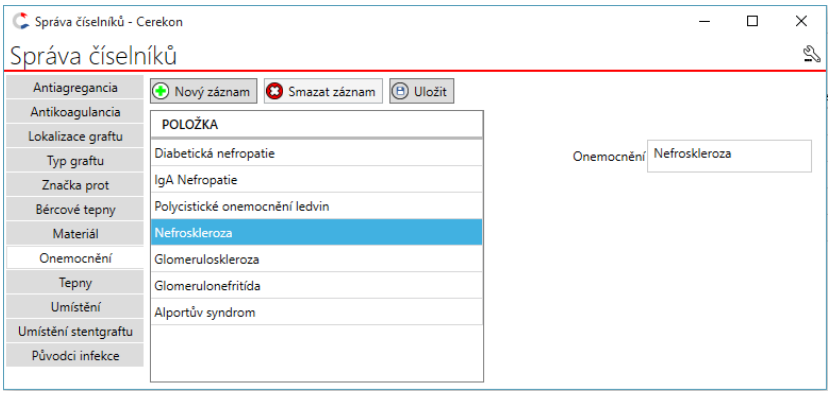
Obrázek B.18: Dialog správy uživatelských rolí.

Informace o připojeních (B.19) zobrazuje seznam uživatelských session.



Obrázek B.19: Seznam otevřených připojení.

Správa číselníků (B.20) umožňuje přidávat nebo editovat číselníkové položky. V seznamu vlevo vyberete číselník a ve střední části se zobrazí hodnoty číselníku, které můžete přidávat nebo upravovat v pravé části okna.



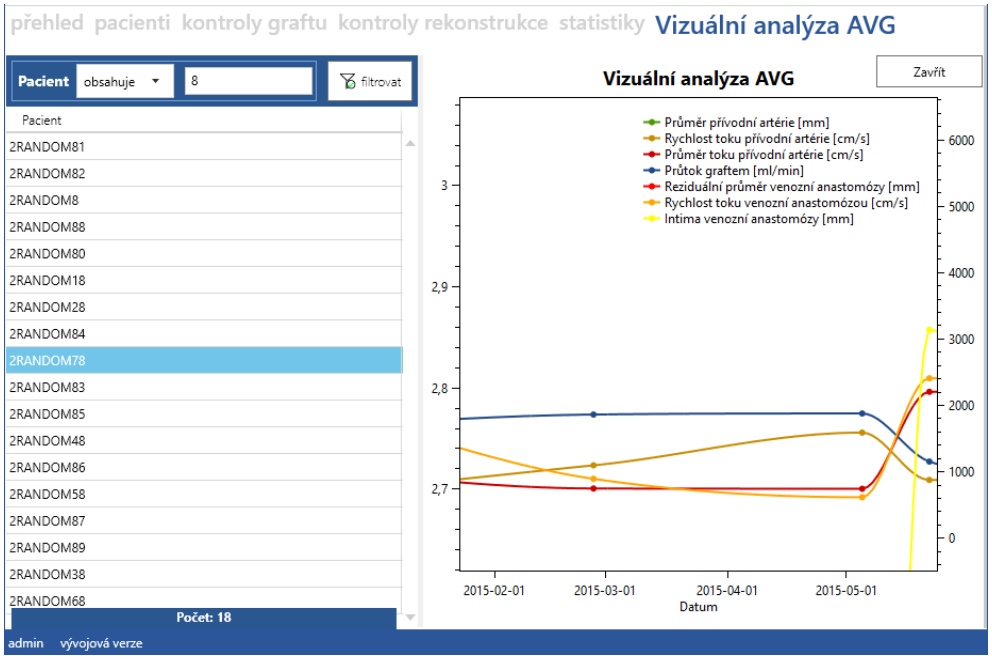
Obrázek B.20: Správa číselníků.

B.7 Rozšiřující moduly

Rozšiřující moduly je třeba ke klientské aplikaci doinstalovat. Instalace probíhá obdobně jako instalace aplikace. Při volbě umístění instalace je potřeba zvolit umístění, kde je nainstalovaná i klientská aplikace. Po instalaci modulu je nutné zkontrolovat konfigurační soubor klientské aplikace `Cerekon.exe.config`. Musí obsahovat cestu k nainstalovanému modulu v sekci `probing/privatePath`. Nicméně instalátor modulu by měl cestu k modulu automaticky přidat.

B.7.1 Vizuální analýza AVG

Modul Vizuální analýza AVG má jednoduché ovládání. V seznamu pacientů vlevo, ve kterém je možné filtrovat, vyberete pacienta, jehož výsledky jsou vizualizovány v grafech v pravé části. Klikem levého tlačítka kurzoru se zobrazí hodnota křivky v daném bodě. Pomocí pravého tlačítka je možné grafy posouvat a pomocí kolečka je přibližovat nebo oddalovat.



Obrázek B.21: Rozhraní modulu Vizuální analýza AVG.

B.7.2 Statistiky

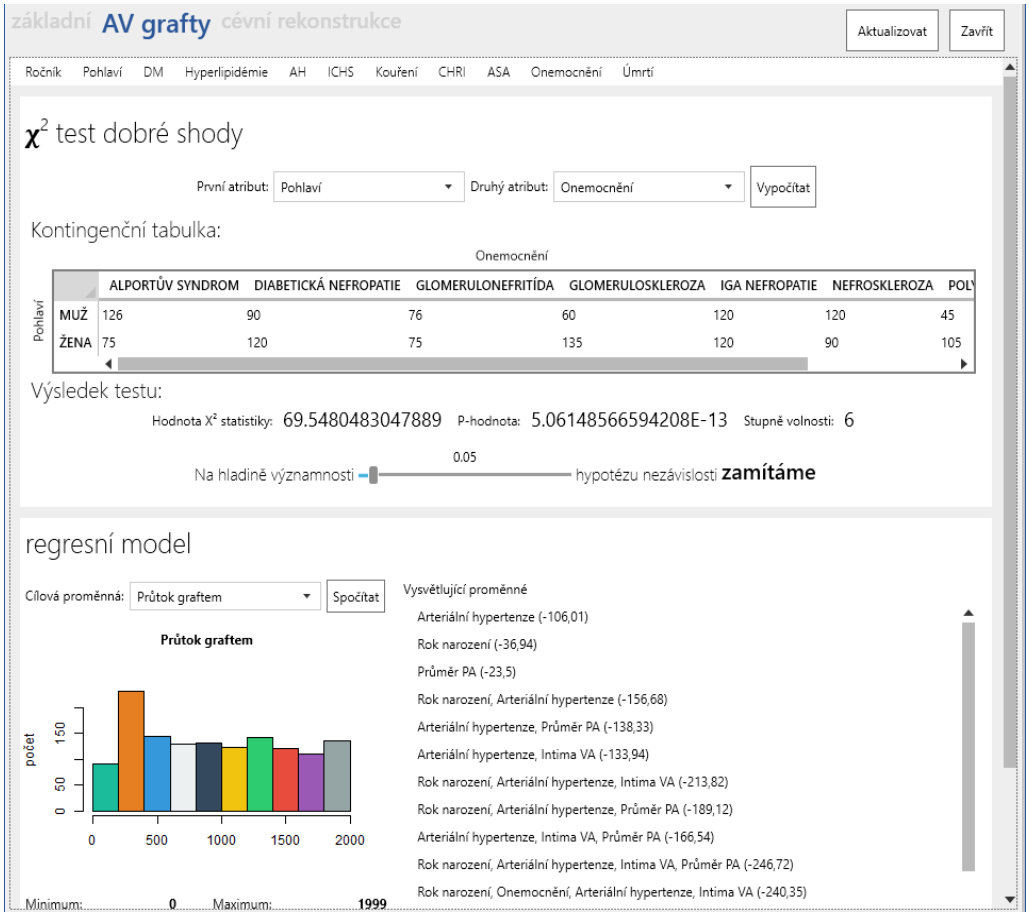
Modul Statistiky má rozhraní rozdělené do dvou částí:

1. základní
2. AV grafty a cévní rekonstrukce

Záložka „základní“ obsahuje grafy zobrazující základní údaje. Při najetí kurzorem na graf se v pravém horním rohu zobrazí tlačítko pro uložení. Po jeho stisku se zobrazí dialog pro uložení grafu jako obrázku. Zároveň se obrázek uloží do systémové schránky (i bez uložení souboru). V horní části je možné pomocí filtrů omezovat kontext. Filtr přidáte stiskem pravého tlačítka kurzoru na příslušný atribut.

Na dalších záložkách se nachází rozhraní pro chí-kvadrát test a pro regresní analýzu. Chí-kvadrát test provedete výběrem dvou atributů z nabídky a stiskem tlačítka „Vypočítat“. Posuvníkem v dolní části můžete měnit hladinu významnosti a tím i výsledek testu.

Pro regresní model nejprve vyberte cílovou proměnnou, kterou potvrdíte stiskem tlačítka „Spočítat“. V seznamu v pravé části se zobrazí uvažované modely. Po výběru modelu ze seznamu se níže zobrazí grafy charakterizující lineární model.



Obrázek B.22: Rozhraní modulu Statistiky.

C. Instalace serveru

V následující kapitole popíšeme kroky potřebné ke správnému nastavení serverové části systému. Jednotlivá nastavení je možné rozdělit do několika kroků:

1. Instalace souborů
2. Konfigurace operačního systému
3. Konfigurace aplikačního serveru
4. Konfigurace databáze

C.1 Instalace souborů

Prvním, a zároveň nejjednodušším, krokem je nainstalování potřebných souborů, tj. knihoven, konfiguračních souborů, hostitelské aplikace a dalších zdrojů. Pro zjednodušení této fáze instalace jsme připravili instalační program. Stačí spustit `CerekonServerInstaller.exe` a řídit se pokyny instalátoru. Instalační balíček byl vytvořen pomocí systému *Inno Setup*[1] a jeho ovládání by tak mělo být intuitivní a nemělo by se značně lišit od jiných instalátorů.

C.2 Konfigurace operačního systému

Protože je komunikace mezi serverovou a klientskou částí systému zabezpečená, je nezbytné mít k dispozici potřebné certifikáty. Nyní popíšeme postup, jak je možné si takové certifikáty jednoduše vygenerovat. Pro tyto účely budeme používat nástroj *MakeCert*[2], který je součástí *Windows SDK*[3].

1. Jako první vytvoříme certifikát pro certifikační autoritu, která bude ručit za všechny další potřebné certifikáty:

   ```
   makecert -n "CN=CerekonCA" -r -pe -sv CerekonCA.pvk
       CerekonCA.cer
   ```

   Tento příkaz vytvoří certifikát s názvem *CN=CerekonCA* (`-n`)[4], který je tzv. *self-signed*[5] (`-r`) a s exportovatelným privátním klíčem (`-pe`). Privátní klíč se uloží do souboru *CerekonCA.pvk* (`-sv`) a celý certifikát do souboru *CerekonCA.cer*.

   Během zpracování příkazu se zobrazí výzva pro zadání hesla k privátnímu klíči. Je potřeba zadat heslo společně s jeho potvrzením a stisknout *"OK"*.

[1] `http://www.jrsoftware.org/isinfo.php` [citováno 11. května 2016]

[2] `https://msdn.microsoft.com/en-us/library/windows/desktop/aa386968(v=vs.85).aspx` [citováno 11. května 2016]

[3] `https://www.microsoft.com/en-us/download/details.aspx?id=8279` [citováno 11. května 2016]

[4] V popiscích příkazů budeme v závorce uvádět příslušný přepínač, ke kterému se popis vztahuje.

[5] `https://en.wikipedia.org/wiki/Self-signed_certificate` [citováno 11. května 2016]

Posléze se zobrazí další výzva, kde je třeba zadat heslo zvolené a zadané v předchozím dialogu. Teprve poté dojde k vygenerování souboru certifikátu.

2. Následně vygenerujeme konkrétní certifikát pro serverovou aplikaci.

```
makecert -iv CerekonCA.pvk -n "CN=127.0.0.1" -ic
   CerekonCA.cer Cerekon.cer -sr LocalMachine -ss My
    -sky exchange
```

   Tento příkaz vytvoří certifikát, za který ručí autorita s privátním klíčem v souboru *CerekonCA.pvk* (`-iv`) a certifikátem v souboru *CerekonCA.cer* (`-ic`), pro jméno *CN=127.0.0.1* (`-n`). Certifikát se uloží do souboru *Cerekon.cer* a nainstaluje do úložiště certifikátů pro počítač (`-sr LocalMachine`) do složky *My* (`-ss`).

   V průběhu je opět potřeba zadat heslo zvolené při generování prvního certifikátu.

3. Nyní je potřeba certifikát připojit k příslušnému portu, na kterém pak serverová aplikace poběží, resp. bude na něm tzv. naslouchat. Pro to použijeme nástroj (prostředí) *netsh*, konkrétně *netsh pro http*[6].

```
netsh http add sslcert ipport=127.0.0.1:29292
   certhash=ed731d209aacf309fc611b08b0aa7bdf7b901898
    appid={24ad2869-e56d-4559-bfaf-14013e7603ec}
```

   Certhash je thumbprint certifikátu. Appid je identifikátor assembly hostitelské serverové aplikace.

   Tento příkaz přidá certifikát (`add sslcert`) pro IP vlastní IP adresu (`127.0.0.1`) a daný port (`29292`), který má uvedený otisk (`certhash=ed731...`) pro aplikaci s daným identifikátorem (`appid={24ad...`).

4. Ve většině případů bude v nastavení firewallu nestandardní port uzavřený. Pro funkčnost systému je tedy nezbytné daný port speciálním pravidlem otevřít.

C.3 Konfigurace aplikačního serveru

V tuto chvíli máme dokončeno nastavení operačního systému, v jehož prostředí poběží aplikační server. Nejen protože je každý vygenerovaný certifikát unikátní, je nezbytné provést změny i v konfiguračním souboru aplikačního serveru. Dále je potřeba provést správné nastavení připojení k databázi, adres a připojení k e-mailovému serveru.

C.3.1 Připojení k databázi

První sekce konfiguračního souboru obsahuje nastavení připojení (pl.) k databázi. První konfigurace připojení, identifikované názvem `CerekonDB` je použita

[6] `https://msdn.microsoft.com/en-us/library/windows/desktop/cc307236(v=vs.85).aspx` [citováno 11. května 2016]

pro hlavní komunikaci aplikace s databází. Do atributu `connectionString` je nutno zadat validní *connection string* pro příslušnou verzi SQL Serveru. Uživatel s ID uvedeným v tomto řetězci by měl mít neomezený přístup k databázi.

```
<add name="CerekonDB" providerName="System.Data.SqlClient"
 ↪  connectionString="Data Source=195.113.19.171;Initial
 ↪  Catalog=CEREKON_PRO; User ID=sa; Password=serverAdminPassword;
 ↪  MultipleActiveResultSets=True"/>
```

Druhá konfigurace připojení, `ReadonlyCerekonDB`, slouží pro komunikaci doplňků (rozšíření, pluginů) s databází. Z bezpečnostních důvodů proto nedoporučujeme použít na tomto místě stejného uživatele jako v předchozím odstavci. Příklad, jaká oprávnění nastavit takovému uživateli, uvedeme v další části věnované konfiguraci databáze.

C.3.2 Nastavení síťových adres

Jako další je zapotřebí nastavit IP adresy, na kterých bude aplikační server naslouchat požadavkům. A to jednak adresu pro hlavní komunikaci klienta se serverem a pak také adresu pro přístup k webovému rozhraní. Obě nastavení jsou atributem `baseAddress` podelementů `add` elementu `baseAddress`.

```
<baseAddresses>
   <add baseAddress="net.tcp://localhost:29292/cerekon" />
   <add baseAddress="http://localhost:29293/cerekon/webpoint"/>
</baseAddresses>
```

Název v adrese pro hlavní komunikaci, tj. pro net.tcp, by měl odpovídat názvu, který byl použit pro vygenerování certifikátu, resp. pro jeho připojení k danému portu. Port musí být také shodný s portem uvedeným při připojování certifikátu.

C.3.3 Nastavení e-mailu

V následující sekci se budeme věnovat nastavení údajů pro připojení k SMTP serveru, který slouží aplikačnímu serveru pro odesílání e-mailů. V následující tabulce vždy pro příslušný klíč - hodnotu atributu `key` podelementu `add` elementu `appSettings` (dále budeme označovat jako "*sekci* ***`appSettings`***"), popíšeme jeho účel.

Klíč	**Popis**
mail_from	E-mailová adresa, která se bude zobrazovat jako odesílatel
mail_server	Adresa e-mailového serveru
mail_port	Port pro komunikaci se SMTP serverem
mail_username	Uživatelské jméno
mail_password	Heslo k e-mailovému účtu

Tabulka C.1: Atributy nastavení e-mailu

C.3.4 Nastavení timeoutů

Uživatelská session má v systému dva timeouty. První, typicky delší, měří dobu neaktivity uživatele. Po jeho uplynutí je uživatel notifikován a zároveň se spustí odpočet druhého timeoutu. Pokud do jeho uplynutí uživatel nevykáže potřebnou aktivitu, je jeho session ukončena. Obě hodnoty se nastavují opět nastavují v `appSettings`.

Klíč	**Popis**
session_inactivity	Čas v sekundách, který slouží pro měření neaktivity uživatele.
session_timeout	Čas v sekundách měřený po uplynutí timeoutu pro neaktivitu.

Tabulka C.2: Atributy nastavení timeoutů

C.3.5 Nastavení Node.js

Pro správu aplikačního serveru je možné využít webové rozhraní, které pro získání dat s aplikačním serverem komunikuje prostřednictvím serveru Node.js[7]. V sekci `appSettings` Je třeba správně nastavit adresu a port Node.js serveru, který se má použít.

Klíč	**Popis**
socket_address	Adresa Node.js serveru.
socket_port	Port, na kterém Node.js server poslouchá.

Tabulka C.3: Atributy nastavení Node.js

C.3.6 Nastavení typu prostředí

Ve stejné sekci, tedy `appSettings`, se nachází i další klíč `runtimeConfiguration`. Ten slouží ke specifikaci, jde-li o instalaci vývojovou, testovací nebo produkční. V tabulce níže uvádíme příslušné hodnoty klíče.

Hodnota	**Popis**
develop	vývojová verze
test	testovací verze
production	produkční verze

Tabulka C.4: Možné hodnoty klíče `runtimeConfiguration`

C.3.7 Nastavení certifikátu

V sekci `behaviors` konfiguračního souboru se nachází element `serviceCertificate`. Jeho atributy musíme upravit právě podle dříve vygenerovaného certifikátu. Tzn. do atributu `findValue` zadat hodnotu jména certifikátu (v příkladu

[7] `https://nodejs.org/en/` [citováno 11. května 2016]

to bylo "127.0.0.1"), do atributu `storeLocation` hodnotu "LocalMachine"a konečně do atributu `storeName` hodnotu "My".

```
<serviceCertificate findValue="127.0.0.1"
↪  storeLocation="LocalMachine" storeName="My"
↪  x509FindType="FindBySubjectName" />
```

C.3.8 Instalace modulu

Serverové části rozšiřujících modulů se instalují typicky pomocí dodaného instalátoru. Potřebné soubory modulu se instalují do složky `Plugins` ve stejném adresáři, jako je nainstalovaný aplikační server. Každý modul zde dále má svou vlastní složku. Nainstalovaný modul, resp. cestu k němu, je třeba zaregistrovat do konfiguračního souboru. Konkrétně cestu k podsložce je třeba přidat oddělenou středníkem k hodnotě atributu `privatePath` elementu `probing`.

```
<runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="Plugins/StatisticsDashboard;
↪  Plugins/AvgAnalysis"/>
    </assemblyBinding>
  </runtime>
```

C.3.9 Konfigurační utilita

Pro zjednodušení konfigurace aplikačního serveru jsme připravili jednoduchý nástroj, který umožňuje komfortněji nastavovat jednotlivé konfigurační hodnoty. Záměrně jej však uvádíme až za popis jednotlivých nastavení, protože bez jejich porozumění by byla konfigurační utilita jen stěží použitelná.

C.4 Konfigurace databáze

Pro další konfiguraci databáze předpokládáme funkční a dostupnou instanci Microsoft SQL Serveru odpovídající všem požadavkům. Dále pak předpokládáme, že uživatel `sa` má plný přístup k prázdné databázi `cerekon`.

C.4.1 Schéma databáze

K vytvoření schématu databáze slouží tři soubory obsahující T-SQL skripty, které v databázi definují potřebné struktury. Skripty je nutno spouštět v daném pořadí. Nicméně může dojít k situaci, kdy je potřeba některé skripty spustit opakovaně, neboť je mezi nimi částečná cyklická závislost. Taková závislost je legitimní a může být například mezi pohledem a funkcí, nebo naopak.

Nejjednodušší spuštění skriptů na databázi SQL Serveru umožňuje nástroj SQL Server Management Studio[8] (SSMS).

[8] `https://msdn.microsoft.com/en-us/library/ms174173(v=sql.120).aspx` [citováno 11. května 2016]

Pořadí	Název souboru	Popis
1	01_CREATE.sql	Vymaže všechny tabulky z databáze a vytvoří tabulky definované ve skriptu
2	02_VIEW.sql	Vytvoří databázové pohledy
3	03_FCT.sql	Vytvoří databázové funkce a procedury

Tabulka C.5: SQL skripty pro vytvoření databázových struktur

C.4.2 Data

Další databázový skript - `09_DATA.sql` - obsahuje příkazy pro vložení dat do některých tabulek. Mezi takové tabulky patří hlavně číselníky, do kterých se vkládají předdefinované hodnoty. Skript také vytvoří uživatele správce aplikace s uživatelským jménem i heslem *"admin"*.

C.4.3 Nastavení SQL Serveru

V rámci databáze zbývá nastavit samotnou instanci SQL Serveru.

Pro rozšiřitelnost aplikace by měl být z bezpečnostních důvodů vytvořen další uživatel. Tento uživatel bude mít ke schématu `dbo`, které vlastní implicitně uživatel `sa` a ve kterém jsou všechny tabulky, přístup pouze pro čtení. Protože by nemožnost zápisu do perzistentního úložiště byla dosti omezující, vyčleníme tomuto uživateli v rámci databáze nové schéma, ke kterému bude mít plný přístup i pro zápis a modifikaci schématu (tj. bude moci vytvářet tabulky, pohledy, atd.).

Potřebná nastavení instance SQL Serveru provádí skript `08_INSTANCE.sql`.