VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

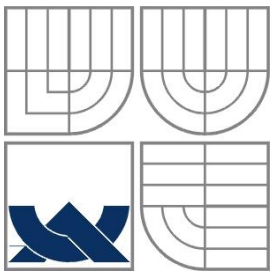
SYSTÉM PRO SPRÁVU DOHOD O ÚROVNI SLUŽEB PODLE ITIL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

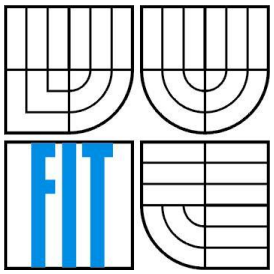
AUTOR PRÁCE
AUTHOR

Bc. RADEK DROZD

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO SPRÁVU DOHOD O ÚROVNI SLUŽEB PODLE ITIL

SYSTEM FOR SERVICE LEVEL AGREEMENT MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Radek Drozd

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. Marek Rychlý, PhD.

BRNO 2014

Abstrakt

Tato práce řeší správu dohod o úrovni služeb podle ITIL ve formě strukturovaného dokumentu na bázi webové služby. Je zde popsán rámec ITIL obecně, následně je provedena analýza, podle rámce definována struktura SLA a na jejím základě definován konceptuální model. Dále jsou prezentovány návrh a implementace webové služby a webového klienta. Obě aplikace budou využity při výuce na naší fakultě.

Abstract

This thesis explores the topic of service level agreements management according to IT Infrastructure Library as a structured document. First of all ITIL in general is briefly described, analysis and a structure of SLA is then defined. This is a base for conceptual model definition. Later design and implementation of a web service and a client web application are presented. Both applications will be used as a tool in the course being held by our faculty.

Klíčová slova

ITIL, IT služba, webová služba, webový informační systém, dohoda o úrovni služeb, hibernate, spring.

Keywords

ITIL, IT service, web service, web information system, service level agreement, hibernate, spring.

Citace

Drozd Radek: Systém pro správu dohod o úrovni služeb podle ITIL, Brno, FIT VUT v Brně, 2014.

Systém pro správu dohod o úrovni služeb podle ITIL

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením RNDr. Marka Rychlého, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Radek Drozd
28. května 2014

Poděkování

Rád bych poděkoval vedoucímu mé práce, RNDr. Marku Rychlému, Ph.D., za nápady, rady a připomínky, které mi během realizace této práce poskytl. Dále děkuji panu Ing. Alešovi Studenému ze společnosti ALVAO s.r.o. za jeho pohled na věc vycházející z praxe.

© Radek Drozd, 2013
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

V současné době je ve světě investováno přibližně 2 500 miliard amerických dolarů ročně do telekomunikačních a IT služeb. Předpokládaný růst za rok 2014 činí přibližně 4%. Nejenom tato čísla dokladující velikost i růstový potenciál trhu s IT službami, ale i snaha udržet si stávající zákazníky a zvýšit náskok před konkurencí vede poskytovatele ke zvyšování kvality poskytovaných služeb. Moderní společnosti se nesnaží znovuobjevovat kolo, proto stále více organizací poskytuje své služby interním i externím zákazníkům na bázi rámce ITIL.

Tato diplomová práce se zabývá popisem, analýzou, návrhem a implementací webové služby určené pro správu dohod o úrovni služeb. Služba uchovává dohody v databázi a poskytuje je ve formě XML dokumentu. Další funkcí služby je porovnání aktuálního stavu služby s hodnotami definovanými v dohodě.

Obecným popisem rámce ITIL se zbývá druhá kapitola, která nejprve popíše rámec jako celek, pak se věnuje životnímu cyklu služby a jeho jednotlivým fázím. Třetí kapitola je věnována specifikaci požadavků na výslednou službu a také na ukázkového klienta. V první části jsou popsány cíle práce a následují nezbytné definice týkající se dohody o úrovni služeb. Hlavní částí je pak definice funkčních a nefunkčních požadavků na aplikaci.

V další části je řešen návrh webové služby a webového klienta. Jako první je uvedena struktura SLA podle doporučení ITIL, na jejímž základu je následně definován konceptuální model. Dále je uvedena série návrhových UML diagramů vztažených jak ke službě, tak k webovému klientu. Pátá kapitola je věnována implementaci výsledných aplikací, popisuje použité technologie, další nabízené možnosti a také proč volil autor právě jejich použití.

V poslední kapitolou je závěr shrnující obsahu textu, za kterým následuje seznam použité literatury.

2 IT Infrastructure Library

Za rok 2013 bylo podle organizace Gartner (americká společnost zabývající se výzkumem a poradenstvím na poli informačních technologií) po celém světě investováno do IT přibližně 3 723 miliard amerických dolarů, z toho 2 581 miliard dolarů bylo investováno do služeb (IT a telekomunikačních). Na rok 2014 společnost Gartner předpovídá růst investic o 4,1% do IT služeb a o 2,3% do telekomunikačních služeb [1].

Z uvedených dat vyplývá, jak horentní sumy jsou do služeb v rámci IT investovány. Zákazníci za své peníze vyžadují dokladovatelnou kvalitu, chtějí vědět, co přesně kupují a kolik je to stojí. Řízením poskytování IT služeb se zabývá manažerské odvětví IT service management (ITSM), definovaný jako *„Implementace a správa kvality služeb IT, které splňují potřeby businessu. Správa služeb IT je vykonávána poskytovateli služeb IT za využití vhodné kombinace lidí, procesů a informačních technologií.“* [2].

2.1 ITIL jako celek

Information Technology Infrastracture Library je poměrně rozsáhlý, volně dostupný rámec zabývající se správou informačních služeb. V současné verzi je tvořen souborem pěti knih vážících se k jednotlivým částem životního cyklu služby. ITIL je úzce propojen s normou ISO/IEC 20000 – mezinárodním standardem pro poskytování služeb. Tato norma z něj vychází. ITIL jako takový však není norma, ale rámec - tedy soubor v praxi dlouhodobě prověřených praktik, obecně popsaných a tím připravených k širšímu použití, nezávisle na konkrétní organizaci. Jednoduše řečeno je hlavním přínosem rámce ITIL jako takového právě tento ucelený soubor, jehož využitím nemusí uživatel znovu vynalézat kolo.

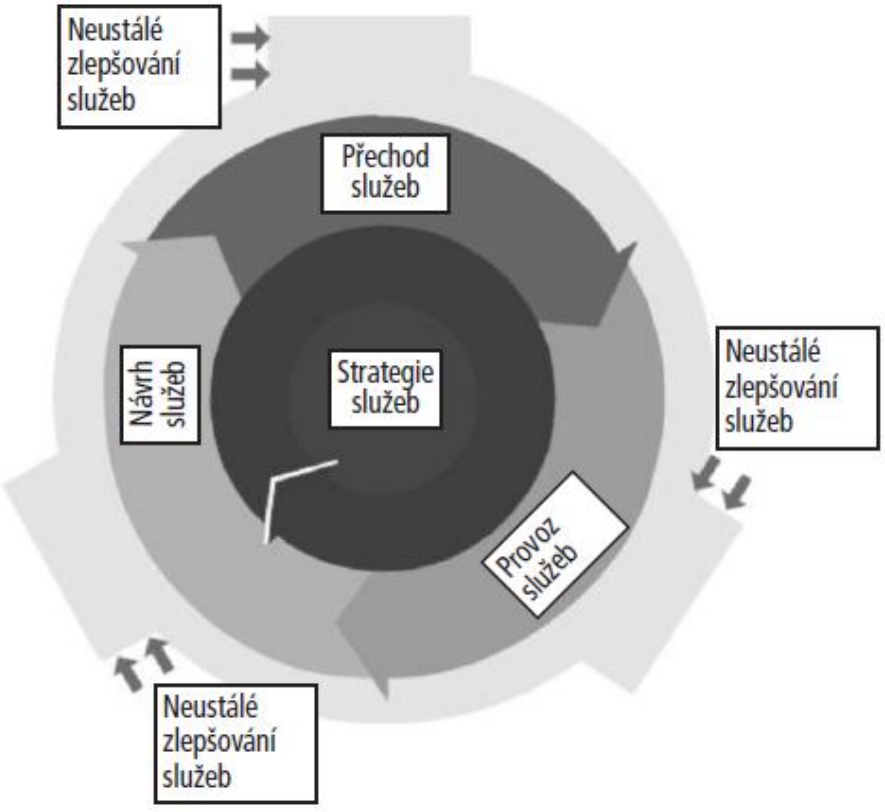
ITIL Začal vznikat již v osmdesátých letech minulého století jako požadavek tehdejší britské vlády vedené Margaret Thatcherovou. Cílem bylo snížit náklady na IT v rámci rozpočtu vlády, což vyústilo ve snahu zprůhlednit a zefektivnit poskytování IT služeb. Neznalost informačních technologií znemožňovala jejich efektivní řízení a nikdo z vedoucích pracovníků jednotlivých ministerstev tak prakticky nevěděl, za co jsou vlastně dané finanční obnosy vynaloženy. Řešením tohoto úkolu byla pověřena agentura CCTA, která výsledky své práce publikovala na konci osmdesátých let ve formě souboru nejlepších praktik pro procesně řízené poskytování IT. Jednalo se vlastně o první veřejnou verzi rámce ITIL. Ten byl následně uzpůsoben pro potřeby průmyslu, čímž vzrostl jeho potenciál pro použití v praxi. V roce 1988 bylo založeno uživatelské fórum, které se postupem času transformovalo do dnešního itSMF (IT Service Management Forum).

Původní verze ITIL – 1.0 OGC čítala 31 svazků a nabídla tak nejobsáhlejší definici procesů vážících se k poskytování IT služeb, která byla kdy zveřejněna. Tento soubor procházel dalším

vývojem, byl provázán s dalšími metodikami. Druhá verze rámce již čítala pouze 7 úzce provázaných publikací. Setřídění a ujasnění obsažených informací umožnilo široké nasazení rámce v praxi. V roce 2007 vyšla poslední edice frameworku – verze 3. V té byl definován životní cyklus služby. Tento soubor byl v roce 2011 dále aktualizován na popud zainteresovaných stran a nese označení *ITIL Edice 2011*. Jedná se zatím o poslední a tedy plně aktuální verzi rámce. Jejím vydáním pozbyla verze 1 svou platnost a bylo doporučeno postupné stažení druhé verze [3].

2.2 Služba a její životní cyklus

Službu ITIL chápe jako *„Prostředek poskytování hodnoty zákazníkovi prostřednictvím výstupů, kterých zákazník chce dosáhnout bez vlastnictví specifických nákladů a rizik. Někdy se termín služba užívá jako synonymum pro klíčovou službu, službu IT nebo balíček služeb.“*[2]. Můžeme si pod tímto pojmem představit jakoukoliv hodnotu spojenou IT dodávanou interními či externímu zákazníkovi interním či externím dodavatelem. Již z definice tak vyplývá, že ITIL je silně procesně orientovaný na zákazníka. S tímto znakem se setkáváme u procesních organizací obecně.



Obrázek 2.1: Životní cyklus služby podle ITILv3. Převzato z [4]

Životní cyklus služby je v rámci ITIL chápán a definován následovně: „*Přístup ke správě služeb IT, jenž zdůrazňuje význam koordinace a řízení napříč různými funkcemi, procesy a systémy, což je nutné pro správu kompletního životního cyklu služeb IT. Přístup životního cyklu uvažuje*

strategii, návrh, přechod, provoz a neustálé zlepšování služeb IT. Je také znám jako správa životního cyklu služby.“ [2]. Model životního cyklu se tedy sestává z 5 fází, z nichž každá bude následně krátce popsána. Můžete jej vidět na obrázku 2.1.

2.2.1 Strategie služeb

První z pěti knih ITIL – strategie služeb se nachází uprostřed modelu životního cyklu. Jak toto umístění naznačuje, procesy strategie se prolínají všemi zbývajícími částmi cyklu. Je zde kladen důraz na celkové zasazení služby do kontextu politiky a cílů organizace. Popisujeme, jak daná služba pomáhá poskytovateli dosáhnout strategických cílů organizace. Výsledkem fáze je plán, který má poskytovatel provést, aby dosáhl strategických cílů organizace. Zahrnuje procesy vytyčení cílů, správu portfolia, správu financí v IT, správu požadavků, vztahů s businessem a v neposlední řadě správu rizik [5].

2.2.2 Návrh služeb

V této fázi je popsáno, jak navrhnout a vytvořit službu, která bude společnosti pomáhat generovat zisk. Současně jsou navrhovány procesy pro provoz služby, již v této fázi jsou vytvářeny a definovány metriky a měřící metody, které slouží jako vstup do procesu neustálého zlepšování služeb. Tyto metriky jsou definovány v rámci procesu správy úrovně služeb, který ústí v definici dohody o úrovni služeb.

Návrh služby se provádí zpravidla formou projektu, který provede sběr požadavků na službu, podle kterých následně navrhne parametry vhodné služby. Nad tímto konceptem se dále vede debata, zda, naplňuje strategické a obchodní cíle organizace, jinými slovy, zda je v souladu se strategií definovanou v předchozí fázi. V rámci této diskuse jsou zhodnocena a definována rizika vztahující se ke službě. Je třeba zmínit, že návrh služby se nutně nemusí vztahovat pouze k tvorbě nové služby, ale popsané procesy mohou být aplikovány rovněž při přepracování, změně stávající služby [6].

2.2.3 Přechod služeb

V této části je snaha vytvořit standardizované rozhraní mezi návrhem a provozem služeb a to pomocí komplexně popsaných procesů vedoucích k minimalizaci nákladů na přechod, dodržení stanovených měřítek služby a to vše ke spokojenosti businessu i zákazníka. Tento soubor doporučených procesů nám umožňuje provádět efektivně a účinně. Patří sem procesy zaměřené na plánování přechodu, testování, vyhodnocení změn, správa jednotlivých vydání, konfigurací a znalostí. Jednou z nejpodstatnějších částí popsaných v této části je správa změn. Tato stanoví přesnou posloupnost činností jednotlivých subjektů nutných k tomu, aby mohla být provedena změna ve službě [3].

2.2.4 Provoz služeb

Z pohledu poskytovatele služby je tato část životního cyklu tou fází, ve které se většinou služba nachází po nejdelší dobu své existence. Jsou zde popsány nejen procesy zabezpečující provoz služby dle dohodnutých parametrů, ale i postupy zaměřené na správu celkových nákladů služby umožňujících určení ceny provozu služby i po několika letech provozu. V této části je také definována funkce *service desk*, která je jediným kontaktním místem mezi poskytovatelem a zákazníkem. Typicky spravuje incidenty a žádosti na změnu. Z pohledu této práce je právě service desk důležitou komponentou, která bude využívat vytvářenou službu v roli posluchače[2].

2.2.5 Neustálé zlepšování služeb

Tato fáze životního cyklu obklopuje všechny ostatní a stará se o to, aby služby vyhovovaly neustále se měnícím parametrům businessu, požadavkům zákazníků a změnám v okolním světě. Tento proces v jádru probíhá podle Demingova cyklu (též PDCA cyklus). Podstatou jsou 4 neustále se opakující fáze (plánuj, dělej, kontroluj a jednej), které vedou k průběžnému zvyšování stupně zralosti poskytování IT služby v čase. Další podstatnou částí rozebíranou v této knize je popis definice metrik, měření kvality služeb a vyvozování závěrů ze získaných dat. Tato část je pro tuto práci poměrně důležitá a je proto metrikám věnována Metriky a monitorování.

3 Specifikace požadavků

Třetí kapitole je věnována specifikaci požadavků na výslednou aplikaci. Tato specifikace je nutná pro setřídění informací, je kritická pro úspěšné dokončení a slouží jako jednotný zdroj informací pro následující návrh aplikace.

3.1 Cíle práce

Účelem tohoto projektu je navrhnout, implementovat a poskytnout volně k použití aplikaci – webovou službu, která bude umožňovat hromadnou správu dohod o úrovni služeb. Jako první je nutné vytvořit konceptuální model dohody o úrovni služeb v souladu s ITIL, který bude využit pro uložení parametrů popisujících konkrétní dohodu. Tento strukturovaný dokument bude pak možné prohlédnout v tisknutelné formě, například ve formátu PDF nebo HTML. Tyto dohody pak služba umožní spravovat - přidání nové dohody, poskytnutí některé existující, úprava. Je nutné vzít v úvahu možnosti dohody orientované na zákazníka a dohody orientované na službu. Dále se jednotlivým požadavkům na správu dohod věnuji v následující kapitole.

Další požadovanou funkcí služby je možnost online monitorování dodržování parametrů SLA, tedy umožnit uživateli provázat konkrétní dohodu nebo skupinu dohod s aktuálními daty o službě a jejich porovnání s limity definovanými v dohodě. Tato část aplikace bude velmi úzce provázána s definicí metrik v dohodě služeb. Aplikace umožní uživateli v rámci definice SLA právě i intuitivní definici těchto metrik. Využití této funkce služby je podmíněno možností zaregistrovat zdroj dat vztahujících se k výkonnostním parametrům služby a stejně tak registraci obsluhy, která má být informována v případě prolomení dohody. Tato komunikace bude probíhat podle definovaného a publikovaného rozhraní. Toto rozhraní je specifikováno ve formě JSON[1], CSV a například XML.

Poslední částí práce je webový informační systém umožňující využití výše specifikované služby. Předpokládaným uživatelem tohoto systému jsou studenti kurzu *návrh a implementace IT služeb* přednášeného v rámci bakalářského studijního programu na naší fakultě. Vedení kurzu umožní přihlášeným studentům přístup k systému. Předpokládá se napojení na fakultní informační systém. Uživateli bude umožněno intuitivně vytvářet SLA podle ITIL, jejichž tvorba je součástí projektu ve zmíněném kurzu. Systém si klade za cíl uživateli poradit, napomoci mu k vytvoření dobře specifikované dohody o úrovni služeb. Jedná se tedy o jistou formu výukového nástroje. Tento informační systém bude implementován jako webový s ohledem na jeho jednodušší nasazení ve školním prostředí. Toto ovšem nijak neomezuje jeho použití mimo fakultu. Systém je implementován na platformě Java EE, stejně tak, jako webová služba. Toto bylo zvoleno z čistě pragmatických důvodů – jednodušší nasazení v prostředí fakulty, multiplatformnost a servery poskytovány zdarma.

[1] Java Script Object Notation

3.2 Dohoda o úrovni služeb

Problematika dohod o úrovni služeb (Service level agreement – SLA) spadá v kontextu rámce ITIL do procesu správy úrovně služeb. Tento proces sleduje požadavky uživatelů kladené na poskytovanou službu, smluvně je ošetřuje. Další součástí procesu je průběžná kontrola a vykazování plnění přislíbené úrovně služeb. Ve vztahu k businessu tvoří tento proces jednotné rozhraní ke všem otázkám spojeným s úrovní poskytovaných služeb. Zahrnuje správu dohodnutých úrovní a správu k ní vztažených dat, které slouží jako důkaz, že tyto vytyčené cílové úrovně byly a jsou dodržovány. V případě prolomení těchto cílů poskytne proces zpětnou vazbu ve formě detailních informací o prolomení dohody.

Dohoda o úrovni služeb je pak jedním z dokumentů vznikajícím v rámci procesu správy úrovní služeb. Jde o psanou dohodu mezi poskytovatelem IT služby a jeho zákazníkem, ve které jsou definovány klíčové cíle a zodpovědnosti jednotlivých zastoupených stran. Zvláštní důraz je kladen na shodu obou stran. Účelem je vybudovat mezi poskytovatelem a zákazníkem partnerství na takové úrovni, že uzavřená dohoda povede ke spokojenosti obou z nich [6].

Variantou dohody o úrovni služeb může být takzvaná *dohoda o úrovni provozních služeb* (Operation level agreement – OLA), která se SLA hodně podobá, ale liší ve smluvních stranách. Tyto dohody jsou totiž uzavírány v rámci různých organizačních celků jedné organizace. Dalším typem smluv, který ITIL zmiňuje, jsou podpůrné smlouvy (underpinning contracts – UC). Ty jsou uzavírány mezi poskytovatelem služby a jeho dodavatelem. Svou strukturou jsou velmi podobné OLA i SLA. Tato práce však cílí pouze na správu dohod SLA.

3.2.1 Typy dohod o úrovni služeb

V rámci ITIL jsou rozlišovány 3 základní druhy dohod o úrovni služeb podle zaměření:

SLA založené na službě – jedná se o dohody pokrývající pouze jednu službu mající stejné parametry pro všechny zákazníky této služby. Taková dohoda je vytvořena v rámci návrhu služby a dále se zákazníkům nepřizpůsobuje na míru.

SLA založené na zákazníkovi – zde se jedná o dohody s individuálním zákazníkem (skupinou zákazníků), které pokrývají všechny využívané služby z portfolia poskytovaných služeb. Zákazníci tento formát dohody upřednostňují právě proto, že pokrývá detailně všechny aspekty poskytovaných služeb v jednom dokumentu.

Více úrovňové dohody – některé společnosti upřednostňují tento formát, neboť umožňuje detailněji rozlišovat mezi výše uvedenými. Často se používají tři úrovně SLA, avšak toto je silně individuální.

Výsledná služba pro správu dohod by tedy měla být připravena na první dvě možnosti a to tak, že bude možné zakládat dohody obou typů. Obsahem dohod o úrovni služeb se zabývá Návrh

aplikace, ve které bude detailně rozebrán konceptuální model. Obecně lze však říci, že v SLA by měly být pouze měřitelné cíle, aby bylo možné jejich následné vyhodnocení.

3.2.2 Rámec SLA

Další z pohledu této práce důležitou částí procesu tvorby SLA je vytvoření rámce SLA. Jde o snahu při správě úrovně služeb vytvořit soubor rámců SLA dohod ke každé dodávané službě a ujistit se tak, že všechny poskytované služby i všichni zákazníci mohou být do systému dohod o úrovních služeb zahrnuti. Tato část je v práci významná především pro navrhovaný editor SLA využívající vytvářenou službu. Je vhodné, aby bylo uživateli umožněno definovat rámec dohody a následně pak vytvářet jednotlivé dohody vázané ke konkrétnímu zákazníkovi odvozením od příslušného rámce [6]. Rámce dohod budou uskladněny na straně služby, ne na straně editoru.

3.2.3 Metriky a monitorování

„Nic by nemělo být zakotveno v SLA, aniž by bylo možné to efektivně monitorovat a běžně měřit.“ Takto v [6] začíná kapitola věnovaná monitorování výkonnosti služby oproti cílům dohodnutým v SLA. Důvodem je dlouholetá zkušenost ukazující, že právě ty pasáže dohod o úrovni služeb, ve kterých jsou zakotveny neměřitelné vlastnosti, se velmi často stávají bodem střetu mezi zákazníkem a poskytovatelem, případně mezi oddělením zodpovědným za SLM a vedením společnosti, které pak typicky postrádá smysl celého SLM. Velký počet společností používá vlastní nebo převzaté informační systémy pro Service Desk, které pak automaticky s využitím logů o událostech slouží i jako zdroj naměřených dat například o času odpovědi na incident.

Monitorování cílů zakotvených SLA velmi úzce souvisí s procesem monitorování dostupnosti služeb. Mezi typické měřené metriky patří *procentuální dostupnost*, *střední doba mezi incidenty*, *střední doba mezi výpadky služby*, *střední doba opravy*. Toto jsou vhodní kandidáti na předpřipravené metriky, které by mohla služby měřit a porovnávat s hodnotami stanovenými v SLA. Takové metriky by měli předdefinované vstupy a byl by popsán i způsob jak je přeměnit v dané výstupy. Popis těchto metrik bude přímo zanesen v rámci SLA, tak aby jej mohla služba dynamicky vyvolat a provést. Dále je předpokládána přítomnost libovolných metrik, u kterých bude mít služby definována vstupní číslo, práh se kterým má být vstup porovnán a místo, kam má být zaslána zpráva o případném překročení prahu.

3.3 Funkční požadavky

V této podkapitole jsou rozebrány požadavky identifikující úkony a aktivity, které musí být nutně vykonány, aby byl projekt tvorby aplikace doveden do zdárného konce. Zde definované požadavky budou primárně využity k definici případů použití ve fázi návrhu. Jsou rozděleny podle toho, ke které

části práce jsou vztaženy do skupiny požadavků na službu a skupiny požadavků na editor dohod o úrovni služeb.

3.3.1 Požadavky na službu

V této kapitole je seznam požadavků na službu, které jsou vždy doplněny krátkým popisem. Pokud není zmíněno jinak, slovo správa obvykle zastupuje operace „vytvoř“, „čti“, „uprav“ a „smaž“ (CRUD z anglického „Create“, „Read“, „Update“ a „Delete“), rovněž tak v následující kapitole.

Správa SLA

Služba pro správu SLA umožní evidovat jednotlivé dohody ve strukturované formě. Jejich příjem i export bude ve formátu XML. Každá dohoda bude jednoznačně identifikovatelná, rovněž bude uchován tvůrce této dohody. Uchovaná dohoda musí být vždy v souladu s definovaným modelem SLA.

Služba bude rozlišovat 3 druhy dohod: SLA, OLA a UC. Dále služba umožní vytvoření rámce SLA dohody. Toto bude umožněno dědičností dohod. Rámec pak bude reprezentován dohodou, ve které nebudou definovány žádné smluvní strany. Tyto budou definovány ve smlouvách, které o této rodičovské dědí. Tím nebude vázán ke konkrétnímu zákazníkovi, ani nebude měřitelný, ale bude sloužit jako vzor pro další dohody. Dohodu bude možné stáhnout ve formě dokumentu (webové stránky) v tisknutelné podobě.

Správa zdrojů informací

Součástí každé dohody jsou měřitelné cíle. Dohoda bez stanovených měřitelných kritérií by měla být službou vyhodnocena jako neúplná – uživatel tedy bude nabádán, aby tyto metriky specifikoval. Konečné rozhodnutí o využití možnosti monitorovat službu pokrytou dohodou však zůstane na jeho uvážení. Jako zdroje vstupů těchto měřitelných cílů mohou figurovat nejrůznější systémy - service desk, sondy zákazníka/poskytovatele, případně měřící systémy třetích stran. Tyto se službou budou komunikovat pomocí definovaného zdokumentovaného rozhraní, které bude specifikováno adresou, na které budou daná data dostupná.

Správa odběratelů

V případě definice metriky v dohodě je možné definovat, zda má být v případě prolomení určeného limitu informována nějaká jiná služba, typicky service desk, případně, zda má být událost pouze zapsána do logovacího souboru. Služba umožní definici více stupňů porušení dohody, mezi kterými je definována relace uspořádání. Odběrové místo informací je pak vázáno ke každému stupni. Je možné specifikovat více odběrových míst k danému stupni poručení dohody.

Export výkazů o plnění cílů

Služba umožní v případě metriky a každého definovaného stupně jejího naplnění (to znamená i pro případ neporušení) nastavit možnost uložení získané hodnoty v daném čase. Tyto logovací soubory musí být z principu neměnné. Z nich pak služba uživateli umožní číst hodnoty o dostupnosti.

Zaslání reportu

Služba umožní uživateli definovat zprávy o metrikách služeb. Jednoduše řečeno uživatel si nastaví, jaké informace chce přijímat (jaké metriky by měly být součástí výkazu) a služba mu pak v definovaný čas zašle souhrn informaci o plnění cílů za období. V základu se budou metriky dělit na ty, které jsou svázány s výkonností služby (performance) a na metriky svázané s dostupností služby (availability). U metrik svázaných s dostupností bude výstupem souhrn klasických metrik s dostupností svázaných.

Vytváření hierarchie dohod

Služba umožní vytvořit svazky dohod pro složitější systémy. Logika věci je vytvořit jednu dohodu o kvalitě služeb, která bude pokrývat dodávku rozsáhlého systému a také umožnit uživateli detailnější specifikaci význačných částí. Například definice dohody o dodávce firemního informačního systému bude obsahovat obecná ustanovení, a bude doplněna dalšími smlouvami pokrývající ERP a CRM část systému.

3.3.2 Požadavky na editor

Druhá část práce – editor dohod o úrovni služeb je prakticky klientskou aplikací implementované v první části služby. Relativně jednoduchý informační systém umožní uživateli jednodušeji pomocí grafického uživatelského rozhraní využívat funkce služby.

Správa uživatelů

Systém správu uživatelů nebude na své straně řešit. Úkolem je systém připravit na připojení na fakultní informační systém. Toto bude realizováno vytvořením vhodného rozhraní, které umožní libovolně měnit autorizační systém. Předpokládá se napojení na fakultní LDAP adresář.

Nastavení systému

Správce systému bude mít možnost nastavení systému. Toto zahrnuje možnost změny adresy a portu služby správy dohod o úrovni služeb.

Správa SLA

Každému uživateli je umožněno spravovat své dohody o úrovni služeb. Tvorba nové dohody bude umožněna tzv. na zelené louce, klonováním stávající dohody, případně vytvořením dohody nad

specifikovaným rámcem. Dohody mohou být samozřejmě upravovány a mazány. Nevyžaduje se uchovávat historii změn dohody. Všechny dohody budou uloženy na straně výše zmíněné služby, Editor zde slouží pouze jako grafické rozhraní.

Důležitou položkou mezi požadavky na klientskou aplikaci je intuitivní použití. Celý editor musí být vytvořen v duchu výukového nástroje. Nepočítá se s jeho nasazením v reálné firmě, ale při výuce předmětu INI na naší fakultě. Studenti by se při použití tohoto klienta měli naučit, jak správně vytvářet dohody o úrovni služeb, které prostředky mohou použít a sami se rozhodnout, zda je použijí.

Současně se správou dohod bude uživateli v rámci každé dohody definovat zdroje vstupů jednotlivých stanovených metrik, stejně tak musí být umožněna definice odběratelů.

3.4 Nefunkční požadavky

V této části budou specifikovány tzv. nefunkční požadavky, tedy ty, které nemají přímý vliv na business stránku aplikace, ale ovlivňují kvalitu aplikace a spokojenost zákazníka při práci s aplikací, její výkonnosti a spolehlivostí. Některé definované požadavky se vztahují oboutm částem práce a nejsou pochopitelně rozepisovány dvakrát.

3.4.1 Požadavky na službu

Jak vyplývá ze zadání, aplikace bude vyvíjena jako webová služba, je třeba dodržet základní principy tvorby aplikací na principu servisně orientované architektury. Dále je potřeba dodržet předepsané formáty pro popis webových služeb.

Dokumentace a štábní kultura

Celá práce by měla být kvalitně zadokumentována pro případ dalšího využití. Jelikož ze zadání vyplývá, že práce musí být publikována jako open source, je kvalitní dokumentace v podstatě nutností. Částečně by jako dokumentace mohla sloužit i tato zpráva, ovšem je třeba vytvořit i návod k použití, aby případný uživatel nebyl zbytečně zatížen kvanty teorie.

Další nutnou položkou vztaženou k dokumentaci je dodržování stanovené štábní kultury pro psaní zdrojových textů. Jako dobrý zdroj informací může posloužit například [7]. Dokumentování zdrojového kódu obou částí bude odpovídat zásadám zvoleného implementačního prostředí.

Open source licence

Jak bylo zmíněno, součástí zadání práce je publikace výsledku online v licenci open source. Jedná se o formu poskytnutí software, kdy jsou nejen k dispozici kompletní zdrojové soubory, ale i o licenci upravující použití software. Více je možné dohledat například na webu *Open source Initiative*[2].

[2] http://opensource.org

Podmínka zveřejnění kódu je v zadání definována pouze u služby, ale na editor se nevztahuje. Je tedy možné, že budou zpřístupněny pouze zdrojové kódy služby a zdrojové kódy editoru nikoliv.

Principy SOA

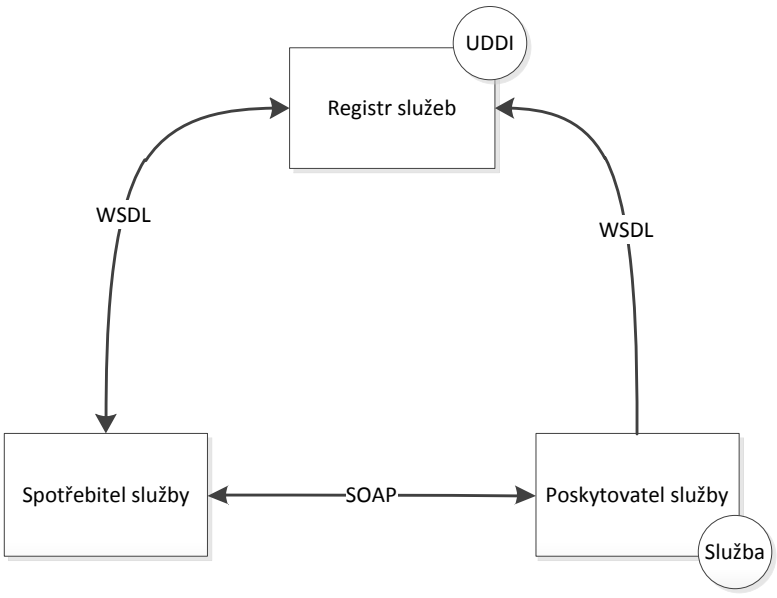
Při návrhu služby je vhodné dodržet 8 základních principů tvorby servisně orientované architektury[3]: standardizovaný kontrakt služby, slabé vazby mezi komponentami, princip abstrakce, znovupoužití, nezávislost, bezstavovost, princip identifikovatelnosti, princip skládání a princip orientace na služby a použití v různých podmínkách. V našem případě je předem jasné, že problémem bude dodržení principu bezstavovosti, jelikož z povahy služby vyplívá nutnost uchování informací o dohodách.

Objektově orientovaný návrh

Při vývoji služby je nutné brát ohled na kvalitně zpracovaný návrh aplikace v souladu s principy OOP. Jako referenční a studijní literatura pro objektově orientovaný návrh posloužila publikace [9] a dále přednášky z předmětu Analýza a návrh informačních systémů na naší fakultě.

Standard webových služeb

Při návrhu a implementaci je nutné dodržet definovaný standard spravovaný konsorciem W3C[4]. Služba a její rozhraní bude tedy popsána protokolem WSDL (Web Service Description Language), s klientem bude komunikovat pomocí textového protokolu SOAP (Simple Object Access Protocol) a bude zapsána v registru služeb. Schéma na obrázku 3.1.



Obrázek 3.1: Architektura Webových služeb podle W3C

[3] Dostupných například zde http://serviceorientation.com
[4] http://www.w3.org/standards/webofservices/

3.4.2 Požadavky na editor

Druhá – rozšiřující část práce je koncipována jako webový informační systém tvořící grafické uživatelské rozhraní pro práci s výše popsanou službou pro správu dohod o úrovni služeb. Následující požadavky jsou tedy z části obecně platné pro většinu webových aplikací.

Intuitivní ovládání aplikace

Aplikace bude figurovat jako grafické uživatelské rozhraní pro ovládání služby. Umožní tak, aby mohl poskytnutou službu využít i běžný uživatel, znalý běžných postupů při ovládání počítače a pohybu na webu. Ovládání služby bez využití tohoto editoru je lidsky takřka nemožné, protože služba komunikuje pomocí textového rozhraní popsaného protokolem SOAP, který pro poměrně obsáhlý na strukturovaný text, jehož tvorba by byla pro uživatele neúměrnou zátěží a použití služby činní prakticky nemožným. Strojově je tento protokol čitelný naopak velmi dobře a knihovny pro jeho zpracování jsou dostupné ve všech zvažovaných programovacích jazycích. Návrh uživatelského rozhraní by se měl řídit obecnými zásadami správného návrhu a použitelnosti.

Webová aplikace

Tento typ aplikace byl zvolen především z důvodu její jednoduché správy a údržby na jednom místě, bez nutnosti instalovat klientský program na uživatelský počítač. Webové aplikace využívají takřka všudypřítomného internetového prohlížeče figurujícího v roli tenkého klienta. S aplikačním serverem pak komunikuje pomocí textového protokolu HTTP. Předpokládá se využití programovacích technologií JSP a vhodného MVC Frameworku. Ke slovu rovněž přijdou základní webové technologie – HTML, CSS a JavaScript.

Více uživatelů

Jak vyplývá z výše zmíněného, aplikace musí umožňovat práci více uživatelům ve stejnou chvíli, jinými slovy přístup k aplikaci musí být neblokující. Stejně tak musí být neblokující použití služby. V případě editoru nebude takový problém tuto podmínku splnit, neboť webové aplikace jsou již ze své povahy více uživatelské.

Objektově orientovaný návrh

Stejně jako v případě webové služby je nutné při návrhu klientské aplikace využít OO návrh. V prostředí webových aplikací je často použit architektonický vzor Model-View Controler, který bude rovněž využit.

4 Návrh aplikace

V této kapitole je v první části rozepsána struktura SLA podle ITIL rámce, na jehož základě je poměrně detailně definován konceptuální model této dohody. Následně je uvedena analýza současných řešení a návrh struktury služby pro správu dohod o úrovni služeb. Struktura je obecně popsána, dále jsou uvedeny diagramy zajímavějších pasáží. Následuje detailní návrh klientské aplikace soustředěný především na přesný návrh posloupnosti jednotlivých obrazovek.

4.1 Konceptuální model SLA

Struktura dohody o úrovni služeb je v rámci ITIL definována ve druhé knize – service design. Je doplněna i o příklad v rámci přílohy. Tato dohoda je psána velmi rozsáhle a spíše ukazuje, co všechno by mohlo eventuálně v rámci SLA zahrnuto, pouze některé části jsou bezpodmínečně nutné. Po nastudování několika příkladných SLA a jejich porovnání se vzorovým příkladem definovaným v ITIL jsem se rozhodl ve své vytvářené službě podporovat komplexní nabídku možných variant zapsaných informací s tím, že naprostá většina z nich bude volitelná (tedy smlouva i bez jejich použití bude validní a systémem přijata) a bude záležet pouze na uživateli, který se rozhodne, zda tu kterou část je výhodné zahrnout do jím tvořené dohody. Toto řešení se mi zdá v souladu s celkovou filozofií ITIL jako rámce, souhrnu nejlepších praktik a doporučení, nikoli striktně definovaného a vyžadovaného standardu, vhodné. Rovněž tak ve službě bude uživatel veden k využití různých částí SLA, ale nebude vyžadováno striktně. V následující podkapitole je popsána struktura dohody o úrovni služeb v rámci ITIL a na ni pak navazuje konceptuální model formou diagramu, doplněný o textový komentář.

4.1.1 Struktura SLA podle ITIL

V této kapitole jsou krátce popsány jednotlivé části dohody o úrovní služeb tak, jak jsou definována v [6], dodatku F. Jak bylo zmíněno výše, jedná se o maximální možný rozsah dohody a některé části tak není nutné použít. Uvádím krátkou sumarizaci informací stěžejních pro tuto práci.

Preambule – hlavička, úvodní část dohody, dalo by se říct, takový abstrakt. Obsahuje základní přehledové informace jako je název dohody, dobu platnosti a stručný popis dohody/služby. V této části jsou rovněž doplněny podpisy zastoupených stran.

Popis služby – komplexnější popis služby, klíčové business funkce, plnění a všechny další relevantní informace popisující službu, její rozsah, dopad a priority pro obchod.

Časová dostupnost služby – popis časového rozsahu, kdy má být služba zákazníkovi dostupná. Typicky ve formátu, od-do, 24/7 a další. Je dobré zmínit podmínky dostupnosti služby (pokud se nejedná o nepřetržitou dodávku) v době svátků, víkendů a tak dále, může být přiložen *kalendář*

služby. Rovněž je třeba dohodnout a smluvně zakotvit časové údaje týkající se údržby služby, v případě, že tato údržba ovlivní její dostupnost zákazníkovi společně s postupy týkajícími se vyjednávání a odsouhlasení případných dalších, smluvně nezakotvených výpadků. Měly by být zmíněny postupy vyjednání trvalé změny časové dostupnosti služby.

Funkčnost – popis detailů poskytované služby, případně počty konkrétních druhů chyb, které jsou v rámci provozu přípustné. Zahrnuje úroveň bezpečnosti a periodu předávání výkazů o fungování služby.

Dostupnost služby – cíle dostupnosti IT služby, kterých by měl poskytovatel dosáhnout v rámci specifikovaných provozních hodin. Obvykle vyjádřeno procenty, periodou měření, metody měření, způsob výpočtu. Všechny tyto části by měly být smluvně podchyceny. Jednotlivé hodnoty mohou být vztaženy k celé službě, podpůrné službě nebo pouze k jejím kritickým částem. Obvykle je velmi těžké vyhodnocovat efektivnost služby formou dostupnosti, proto se používá měření nedostupnosti služby. Z tohoto důvodu se ve smlouvě spíše vyskytuje její definice. Toto funguje za předpokladu, že službě je po většinu času provozu dostupná, pouze výjimečně nedostupná. Jednodušeji řečeno je třeba dohodnou, co bude měřeno, v jakém bodě, s jakou periodou a kdo bude mít měření na starosti. Je možné přenechat měření například třetí straně, pokud se na tom odběratel a dodavatel služby dohodnou.

Spolehlivost – maximální počet možných tolerovaných výpadků za dohodnuté období. Může být vyjádřeno jako počet výpadků, střední doba mezi poruchami (Mean Time Between Failures – MTBF) nebo střední doba mezi incidenty služby (Mean Time Between Service Incidents - MTBSI). Je vhodné doplnit definici výpadku, jak budou výpadky monitorovány a měřeny.

Výkonnost služby – detailní popis očekávané době odezvy IT služby (například průměrná doba odezvy, maximální přípustná doba odezvy, někdy definováno jako procentuálně „99% času bude doba odezvy nižší než 5s“), propustnost služby a další. Vždy doplněno o daný práh. Může obsahovat i počty souběžných transakcí, současně připojených uživatelů a další.

Kontinuita služby – krátká zmínka o plánech kontinuity organizace poskytovatele, společně s detailními informacemi, jak může být SLA zasažena v případě neočekávané katastrofy.

Bezpečnost – krátká zmínka o bezpečnostní politice organizace s odkazem na kompletní dokumentaci bezpečnostní politiky organizace. V případě výjimek a/nebo nadstandardních požadavků je na místě detailní popis.

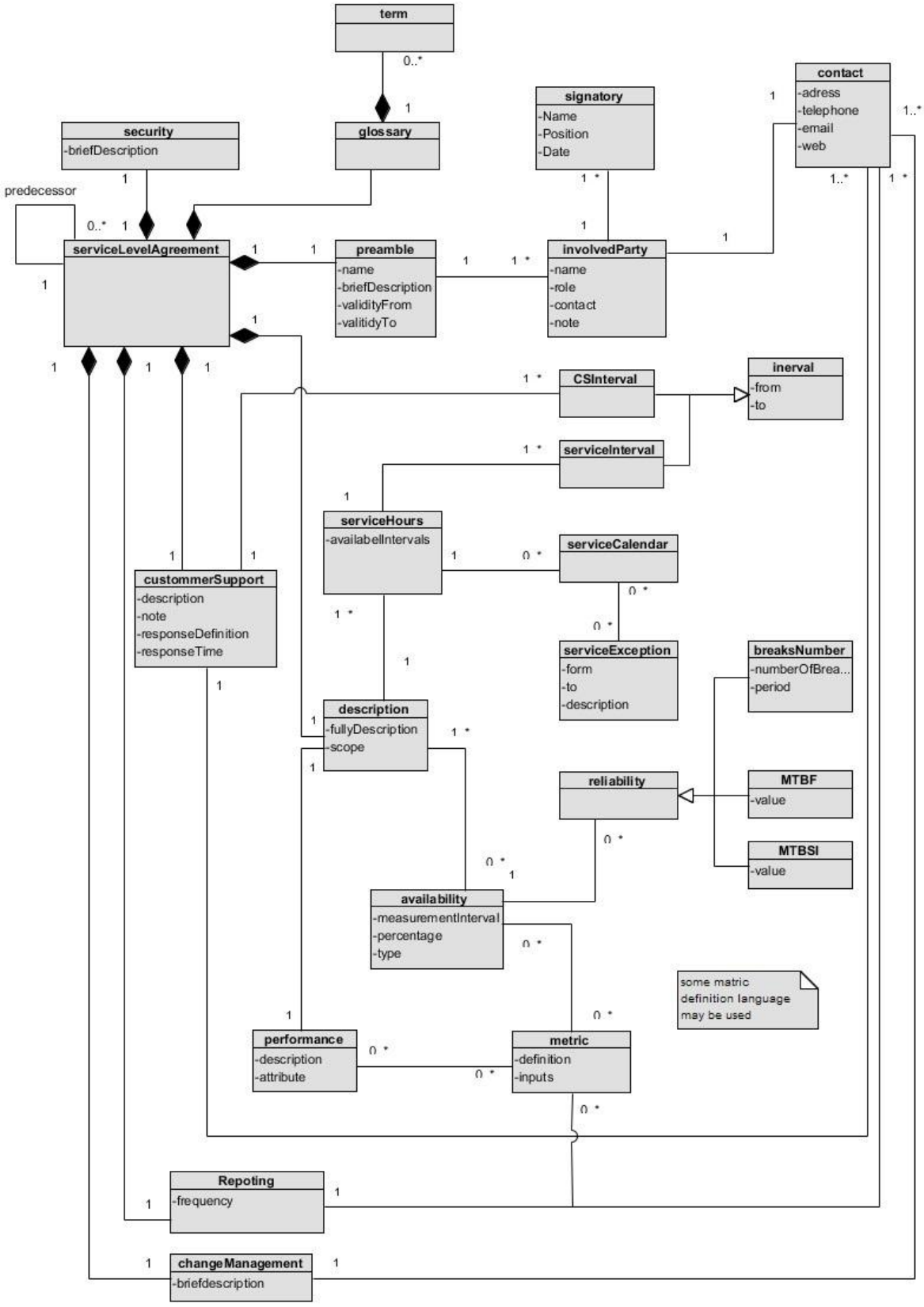
Zákaznická podpora – je třeba zadokumentovat popis, jak a kdy je možné kontaktovat service desk, co dělat v případě, kdy je service desk nedostupný (například podpora třetí strany). Dále je možné definovat přístup k automatické podpoře (znalostní bázi), je vhodné zakotvit i metriky vztahující se k service desku – počet zazvonění telefonu. Doporučuje se zahrnout klíčové časy jako čas odpovědi na incident společně s definicí toho, co chápeme jako odpověď, rovněž tak i limity pro řešení incidentu. Někdy je vhodné doplnit i odkaz na dohody se třetími stranami případně na OLA dohody související s danou službou.

Kontakt – kontakty na obě smluvní strany včetně kontaktního místa pro stížnosti a jejich případnou eskalaci.
Správa změn – popis procedur, případně odkaz na dokument popisující politiku změn týkajících se dohody.
Plnění – pokud je to nutné, je vhodné zmínit detaily týkající se plnění, platební postupy, formuláře, případně další dokumenty. Zde mohou být rovněž zmíněny různé finanční postihy nebo bonusy vztahující se ke sjednávané službě. Kdy na ně vzniká nárok, jak budou vypočítány a zaplaceny. Je nutné zmínit, že finanční postihy mohou být v některých případech nevhodné a vytvářet bariéru ve vzájemné důvěře poskytovatele a odběratele.
Výkazy o fungování služby – definice toho, co všechno bude součástí výkazu o fungování služby, jak často, kým a pro koho budou tyto výkazy vytvářeny. Je nutné kontakt na zodpovědné osoby na obou smluvních stranách.
Slovník – Pokud je ve smlouvě použita speciální terminologie, je vhodné ji sjednotit použitím slovníku.
Dodatek – v případě dodatečných dokumentů nebo příloh je vhodné uvést jejich seznam a dostupnost.

4.1.2 Výsledný konceptuální model

Výsledný konceptuální model je ukázán na obrázku 4.1. Z větší části se jedná o popisné entity jako je preambule, popis managementu změn. Jde vidět zpětná vazba dohody nazvaná *predecessor* – tedy předchůdce. Ta slouží k vytvoření hierarchie mezi dohodami Vícekrát využívanou entitou je kontakt, kterým jsou pokryty všechny zainteresované strany/osoby – jak na úvodní části smlouvy na podepisující zástupce, zainteresované strany, tak na zákaznickou podporu, management změn i jako příjemce pravidelných výkazů.

Další entitou, která stojí za zmínku, jsou metriky. Tyto jsou navázány k dostupnosti služby, stejně tak mohou být použity i k popisu výkonnosti služby a jsou součástí výkazů výkonnosti.



Obrázek 4.1: Konceptuální model SLA

4.2 Služba hromadné správy SLA

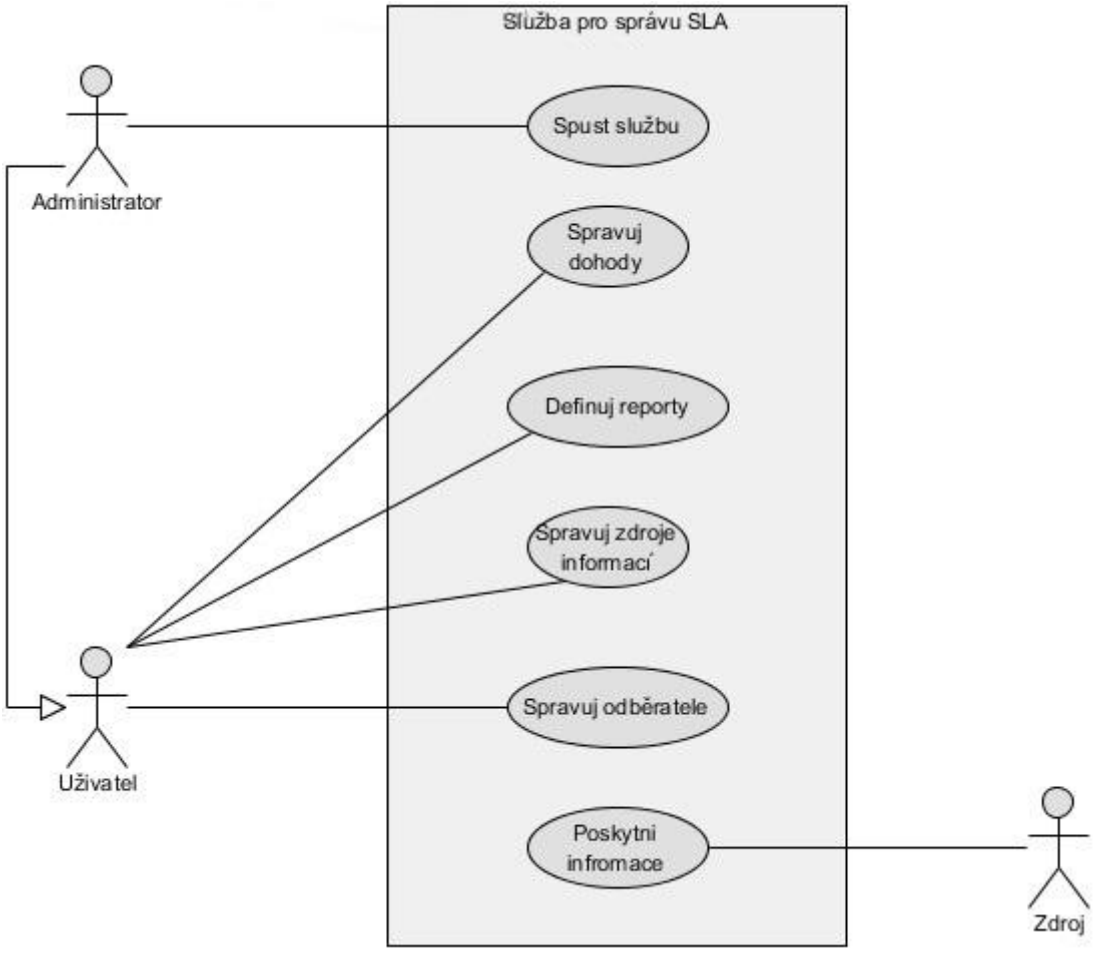
V této kapitole bude předveden návrh výsledné služby ve formě komentovaných diagramů, tak jak vyplynuly z požadavků definovaných v předchozích kapitolách. Bude se jednat o malou jednoúčelovou aplikaci, která by měla být optimalizována na vyšší výkon.

4.2.1 Existující řešení

V současné době existuje na trhu několik různých aplikací, které se zabývají správou dohod o úrovni služeb. Obvykle se ale jedná o rozsáhlejší aplikace, zabývající se kompletní podporou pro řízení business procesů pro poskytování služeb na bázi ITIL, spojené velmi často například s aplikací podporující service desk. Například produkt Service Desk Plus[5] společnosti ManageEngine má podporu SLA zabudovanou. Rovněž společnost BMC Software[6] má ve svých produktech zabudovanou podporu správy a monitoringu dodržování SLA dohod, tentokrát ve shodě s rámcem ITIL. Další produkty řešící monitorování SLA jsou od společnosti Solarwinds[7], která je spojuje dohromady se svými monitorovacími systémy. Zajímavostí je, že tato společnost má pobočku i v Brně. Z velkých společností, které prodávají své proprietární řešení poskytování služeb, je možné zmínit produkt společnosti Dell[8]. Zástupcem volně dostupného software může být například itop - ITSM & CMDB OpenSource[9], který poskytuje kompletní konfigurační databázi, včetně podpory service desku, service managementu, management změn a právě i automatickou správu dohod SLA.

[5] http://www.manageengine.com/products/service-desk/sla-management.html
[6] http://www.bmc.com/products/product-listing/itil-service-level-management.html
[7] http://www.solarwinds.com
[8] http://www.quest.com/foglight-for-service-level-management/
[9] http://sourceforge.net/projects/itop/

4.2.2 Diagram případů použití



Obrázek 4.2: Diagram případů použití služby

Diagram případů použití zachycuje funkčnost systému, primárním cílem je definovat hranice systému a velmi základně odhadnout rozsah systému. V diagramu jsem záměrně pro zvýšení přehlednosti neuváděl CURD akce, které jsou vždy zahrnuty pod jedním případem použití.

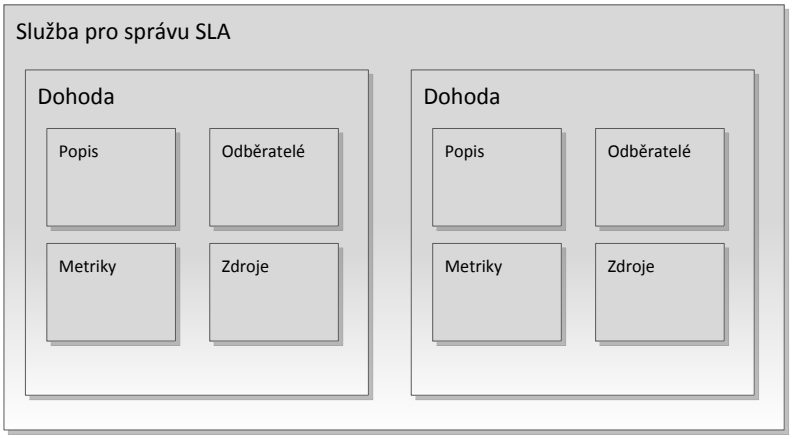
V systému byly identifikovány dvě role uživatelů. Jako první je administrátor, který spravuje běh služby. V rámci tohoto případu užití je možné v konfiguračním skriptu před spuštěním služby nastavit potřebné adresy, službu spustit, vypnout, restartovat.

Další rolí je uživatel služby. Tento má umožněno vše potřebné ke správě a monitorování dohod. Správa zdrojů informaci a odběratelů je vždy vztažena ke konkrétní dohodě, případně je možné je navázat i na rámec dohody. V tomto případě jsou pak tyto spojení defaultně přiřazeny i ke smlouvám odvozeným od daného rámce.

Posledním účastníkem je zdroj informací, který má možnost zaslat (aktuální) informace o specifikované metrice, kterou systém porovná se stanoveným prahem a zachová se podle specifikovaného postupu.

4.2.3 Návrh struktury dat

Z výše zmíněných požadavků jsem navrhl základní strukturu dat. Je třeba, aby aplikace zpracovávala neomezené množství dohod, které budou existovat paralelně, nezávisle na sobě. Každá z těchto dohod bude obsahovat vlastní popis, seznam definovaných metrik, seznam odběratelů provázaný s určitým stavem metriky a seznam zdrojů dat pro porovnání s metrikami. Jediné propojení je možné u rámců dohod a smluv od nich odvozených. V případě úpravy rámce bude uživateli nabídnuto zvolit si, zda si přeje upravit i dohody od něj odvozené, nebo zda má být vytvořena nezávislá verze dohody. Ilustraci můžete vidět na obrázku 4.3.



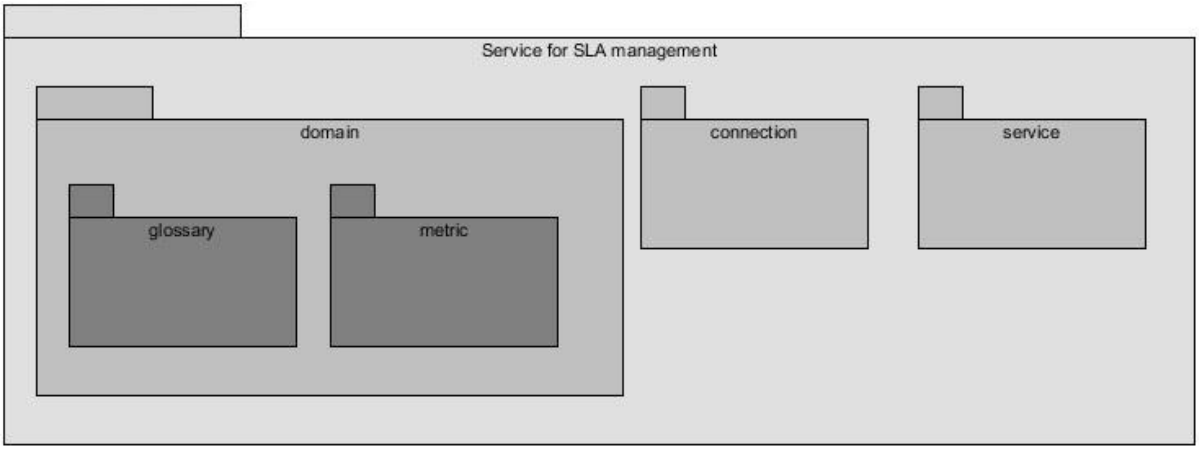
Obrázek 4.3: Schéma struktury dat

4.2.4 Konceptuální model

Služba jako taková tedy bude zahrnovat čtyři hlavní oblasti své činnosti:

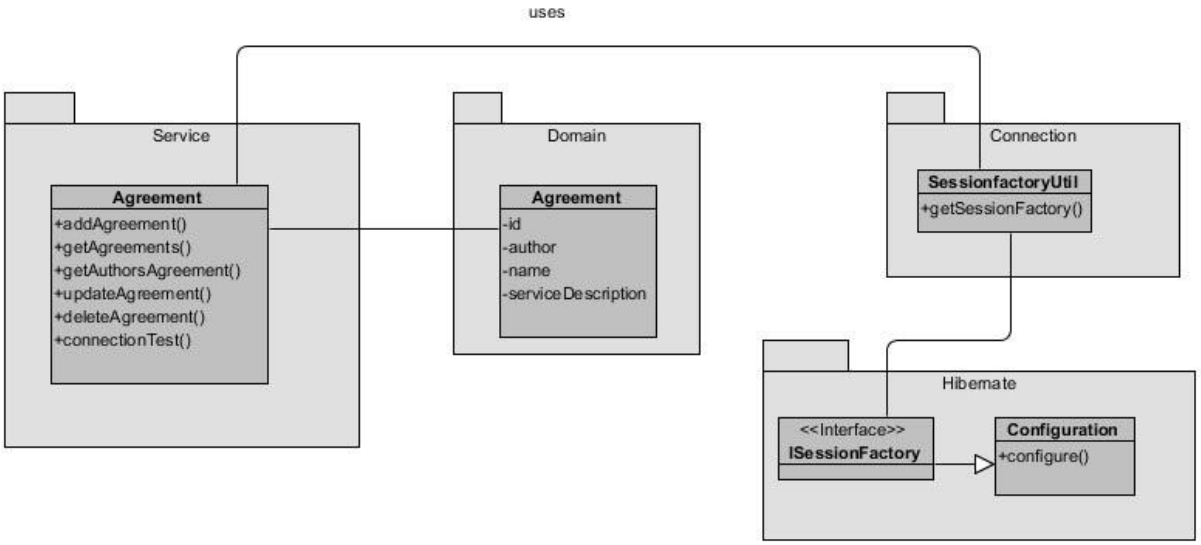
- *Služba* – V tomto balíčku se nachází třída samotné služby. Tato umožňuje uživateli připojit se ke službě a jsou zde obsaženy všechny poskytované metody. Jedná se primárně o metody spojené se správou dohod.
- *Spojení* – V tomto balíčku je proxy třída určená pro komunikaci s databází. Ke spojení s databází a objektově relační mapování je využit Framework hibrnate. Blíže se mu budeme věnovat dále.
- *Doména* – V tomto balíčku jsou obsaženy všechny třídy reprezentující dohodu jako takovou. Jedná se o třídy, které jsou pomocí hibernate mapovány na databázi. Do zvláštních balíčků jsou vyčleněny třídy týkající se definice slovníku a dále třídy spojené s metrikami. Oboje z důvodu možnosti použití balíčku jako samostatného celku (například pro případ návrhu modelu slovníku v případě tvorby jiné služby).

Schéma můžete vidět na obrázku 4.4.



Obrázek 4.4: Diagram balíčků - struktura služby

Služba



Obrázek 4.5: Konceptuální diagram tříd v balíčku správy dohod

V tomto balíčku je pro běh služby umístěna vůbec nejdůležitější třída. Ta slouží jako styčný bod mezi službou a okolím, kterému poskytuje veškeré své metody. Patří sem poskytnutí dohody, možnost dohodu uložit, smazat a upravit. Tato služba využívá pro svou činnost ostatní specifikované třídy – napojení na databázi, za které zodpovídá, i celou doménu. Detailněji je struktura balíčků ukázána na obrázku 4.5. Jelikož služba poběží na aplikačním serveru, není třeba řešit zpracování požadavků ve zvláštních vláknech, to obstará server za nás. Důležité je zmínit využití Hibernate frameworku, který zajistí OR mapování.

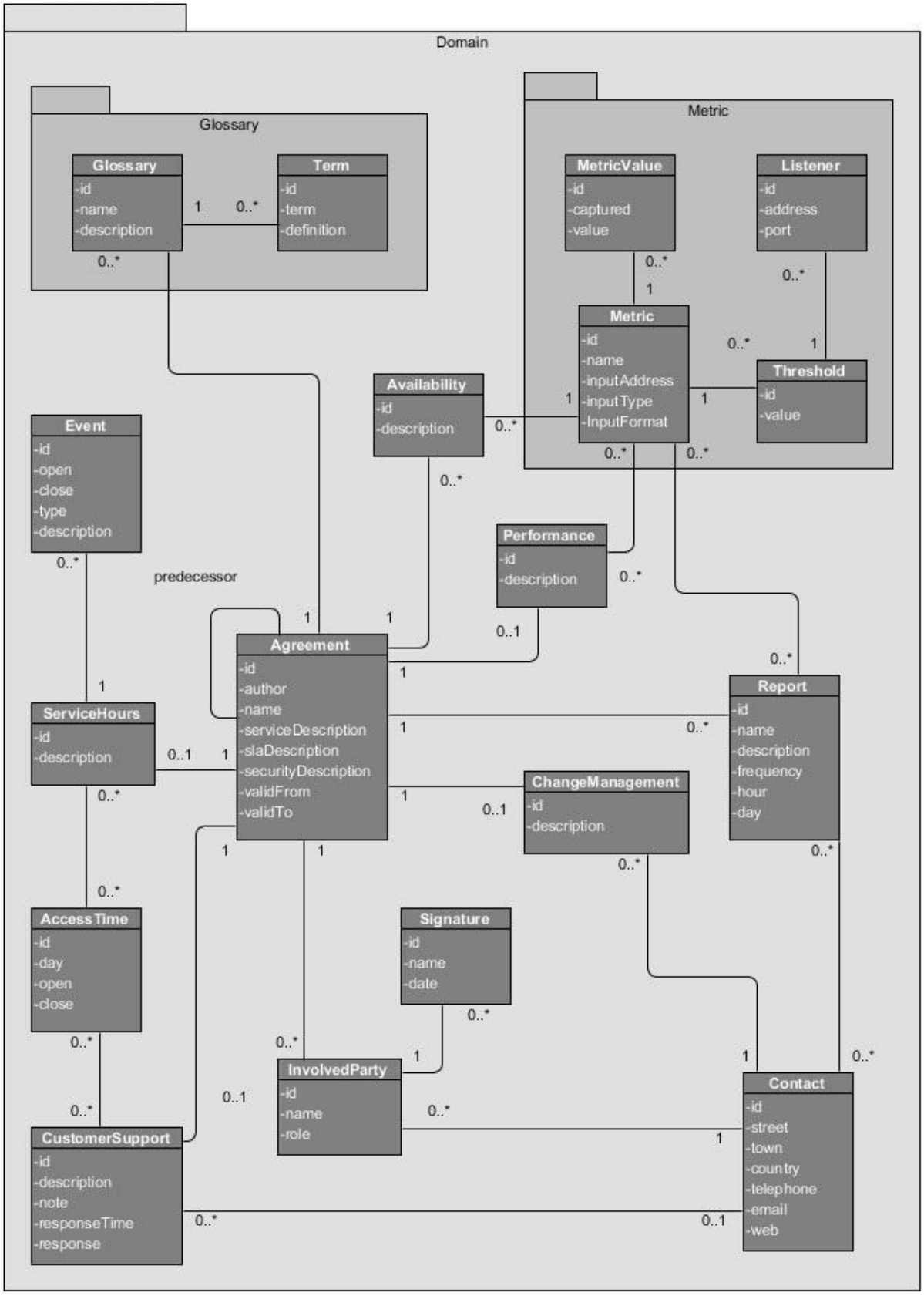
Doména

Doménový diagram vychází z výše uvedeného konceptuálního diagramu. Jak bylo zmíněno, nachází se celý v balíčku `domain` a celkem obsahuje 18 tříd ve dvou podbalíčcích. Hlavní třída opět reprezentuje dohodu jako celek. Nachází se v ní základní informace, jako je vlastník dohody, platnost, popis dohody, popis služby a popis bezpečnosti bezpečnostních pravidel.

Na tuto třídu pak navazují další třídy, nutné pro sestavení dohody. Patří sem například servisní hodiny, případně zákaznická podpora. Obě třídy umožňují blíže popsat danou oblast. V obou třídách figuruje položka specifikující přístupové časy – časy kdy je služba nebo zákaznická podpora dostupná. Další třídou, která se váže k dostupnosti služby, je třída `Event`. Ta slouží k definici výjimek z provozu – různé plánované odstávky a tak dále. Jedná se tedy o reprezentaci kalendáře služby. Rovněž třída věnovaná změnovému řízení je spíše popisnou. U zákaznické podpory bych zmínil zvláště důležitou vazbu na třídu `Contact`. Tato vazba slouží k uchování takzvaného jednotného místa kontaktu, anglicky *single point of contact*. Toto poskytuje jednotný konzistentní způsob komunikace s organizací poskytující službu. Může se jednat například o *service desk* [2].

Třída `Contact` je rovněž využita při ukládání informací o smluvních stranách. Těch může být v dohodě obsažen libovolný počet. Každá z těchto stran má svou roli (zákazník, dodavatel, poskytovatel) a libovolný počet podpisujících. Kontakt je u každé smluvní strany uveden právě jeden, opět v souladu s definicí jednotného místa kontaktu.

První ze dvou balíčků uvnitř domény je slovník. Ten slouží k ujasnění pojmů obsažených v dohodě, aby nedocházelo zbytečně k jejich mylnému výkladu jednotlivých smluvních stran. Druhý vyčleněný balíček obsahuje všechny třídy vázané k metrikám služby. Každá metrika je definována svým názvem, vstupním formátem a vstupní adresou. Na tuto adresu si pak systém „sáhne“ pro data o metrice. K metrice se váže definovaný práh. Těchto prahů může být libovolný počet, což umožňuje vytvářet posloupnost těchto prahů (například, když dojde k mírnému prolomení cílů metriky, je důležitá správa pro service desk, pokud se prolomení stupňuje, je třeba kontaktovat management, atd.). Každý z těchto prahů může být spojen s libovolným počtem posluchačů – tedy míst, které je třeba kontaktovat v případě prolomení prahu. Oba zmíněné balíčky byly vyčleněny, neboť je možné jejich použití odděleně od tříd pro definici dohody. Oba respektují zásadu slabé provázanosti – tedy snahu snížit provázání balíčků na minimum podle GRASP [9].



Obrázek 4.6: Model domény.

Metriky jsou s dohodou svázány prostřednictvím tříd `Availability` (dostupnost), `Performance` (výkonnost) a `Report`. Dostupnost je vždy svázána pouze s jednou metrikou – ta sbírá záznamy o nedostupnosti služby. Je však možné definovat více popisů dostupnosti služby. Správné použití je například v případě, že služba se skládá z několika dílčích částí - v případě webového hostingu by mohlo jít o dostupnost databázové serveru, emailového serveru a samotného aplikačního serveru. Výkonnost naopak může být v rámci smlouvy popsána jen jedna, ale na rozdíl od dostupnosti může být svázána s libovolným počtem metrik.

Metriky definované v obou třídách (dostupnosti a výkonnost) jsou pak využity ve třídě `Report`. Zde uživatel definuje kromě názvu, popisu reportu a informací k zaslání i jednotlivé metriky – vybere si z již definovaných. Tyto jsou mu pak v rámci reportu vyčísleny. Poslední položkou je kontakt, tedy informace kam má být report odeslán.

4.2.5 Databázová vrstva

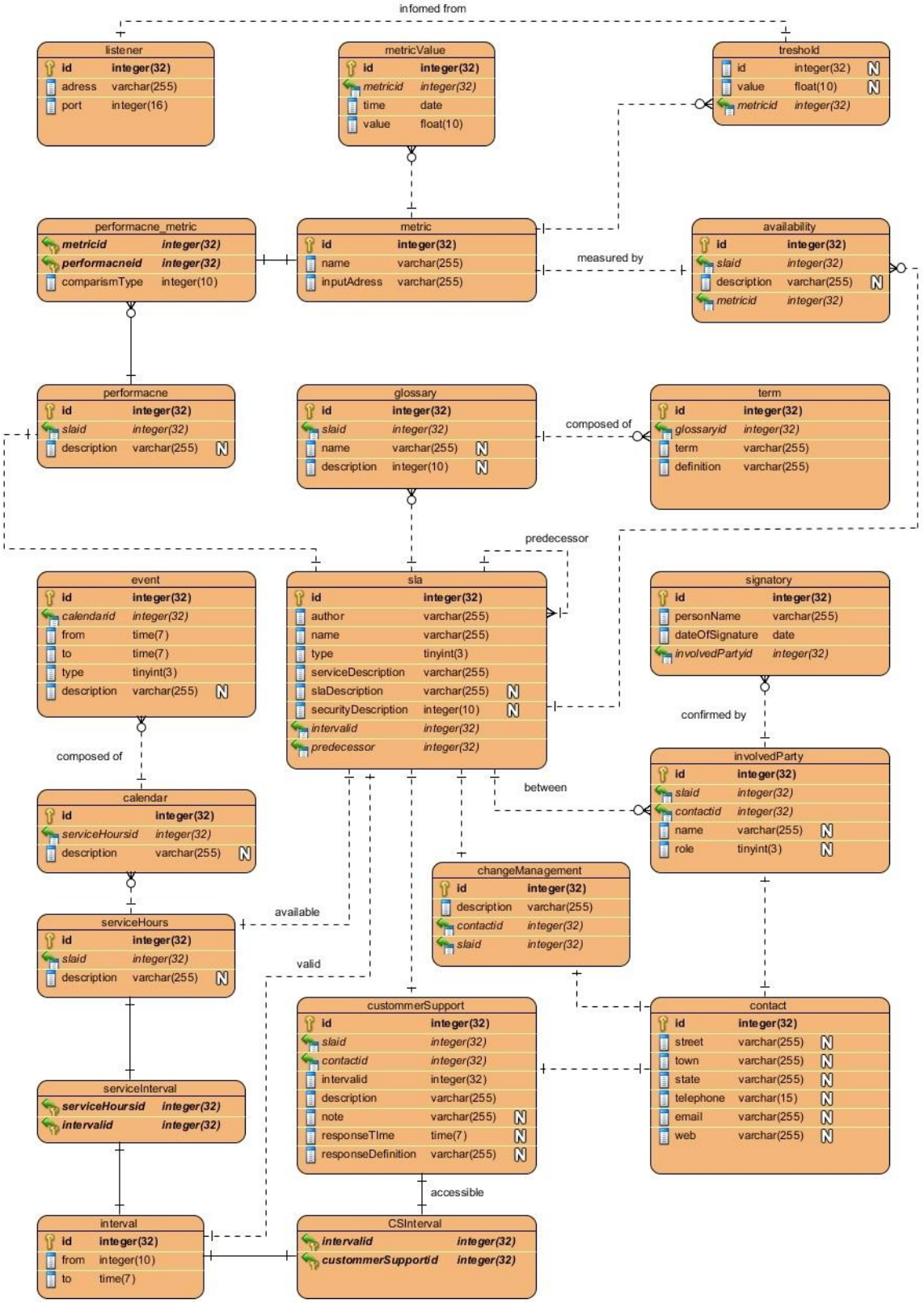
Pro fyzické modelování databázové struktury, nad kterou bude aplikace operovat, byl zvolen ER diagram. Z tohoto diagramu návrh databáze vychází. Při jeho tvorbě pak došlo pouze ke změnám detailů.

Středem a hlavním bodem celého diagramu je samotná dohoda, která má být uchována. Ta obsahuje základní informace popisující dohodu, a které jsem shledal nevhodné modelovat jako samostatné entity. Za zmínku stojí autor, který je uložen pouze ve formě textového identifikátoru – správa uživatelů nebude ve službě obsažena. V diagramu jde vidět zpětná vazba umožňující vytvořit stromovou strukturu dohod. Na dohodu je pak navázáno několik entitních množin ve formě vztahu 1 ku 1. Tyto entity by bylo možné zahrnout všechny do jedné, ale tím by se stal diagram velmi nepřehledným. Z dalších entitních množin zmíním jen ty význačné nebo něčím zajímavé.

- *Kontakt* – univerzální množina, která je spojena jak se zainteresovanými stranami, tak třeba se zákaznickou podporou.
- *Interval* – slouží nám k popisu časového úseku. Jeho pomocí definuje dostupnost služby a dostupnost zákaznické podpory. Rovněž určuje dobu platnosti smlouvy.
- *Metrika* – tato entita definována adresou zdroje a názvem. Může být navázána jak na výkonnost, tak na dostupnost služby. S metrikou je spojeno až několik prahů, z nichž každý může v případě překročení kontaktovat 0 až několik příjemců upozornění.

Diagram se může zdát velmi podobný modelu domény, uvedeném na obrázku 4.6, což je logické, neboť z něj vychází. Jedná se ale o dvě odlišné věci – model domény ukazuje třídy, tak jak budou chápány z pohledu programovacího jazyka. ER diagram na obrázku 4.7 zachycuje databázovou vrstvu aplikace, na kterou budou jednotlivé třídy z modelu domény mapovány. Jde vidět doplnění

o vazební tabulky a současně o položky `idx` – jedná se indexy položek, které jsou uloženy v polích. Důvod bude více rozebrán v kapitole věnované implementaci služby.



Obrázek 4.7: Návrh databáze služby

4.2.6 Vstupní a výstupní protokol metrik

Poslední částí návrhu webové služby je definice komunikačního protokolu, pomocí kterého budou služba získávat/odesílat informace o metrikách. Tyto informace budou textového charakteru, pro přenos bude využit textový protokol aplikační vrstvy – protokol HTTP.

Akceptované formáty

Aplikace bude akceptovat 3 základní textové formáty dat. Prvním z nich je CSV (zkratka z anglického Comma-separated values, tedy hodnoty oddělené čárkou). Tento formát vznikl již v době před osobními počítači, v roce 1967 jej dokázal zpracovat překladač jazyka Fortran. V minulosti byl často používán pro ukládání dat v jednoduchých databázových systémech. V současné době jej akceptují například tabulkové procesory v kancelářských balících MS Office nebo Libre office. Formát vyniká svou jednoduchostí a velmi nízkou režií. Velmi často se využívá pro přenos tabulkových dat, která jsou reprezentovány velmi intuitivně – každý řádek tabulky je vyjádřen jedním řádkem v textovém souboru. Jednotlivé buňky na řádku jsou pak v textovém souboru odděleny předem dohodnutým oddělovačem. Nejčastěji se používají čárka nebo například středník. Pro formát CSV neexistuje plná specifikace, je však popsán v RFC 4180[10].

Druhým akceptovaným formátem je XML (eXtetensible Markup Language – rozšiřitelný značkovací jazyk). Jedná se rovněž o textový formát, derivát složitějšího jazyka SGML. Jazyk XML byl navržen a je dále udržován konsorciem W3C[11] jako metajazyk pro výměnu dat mezi aplikacemi, případně pro publikování dat. Velmi často se využívá pro serializaci dat. Jazyk je zaměřený na obsah – soustředí se výhradně na informace v dokumentu obsažené, neřeší jejich prezentaci jako HTML (pro prezentaci dat popsaných v XML slouží například CSS, kde dodefinujeme, jak má který element vypadat, případně je možné pomocí XSLT transformovat XML dokument do HTML, jehož základní grafická podoba je definována). I přes to však běžný XML dokument obsahuje z 40% značky.

Posledním formátem, který bude služba zpracovávat je JSON (JavaScript Object Notaion). I když se to podle názvu nezdá, jedná se o formát nezávislý na platformě i programovacím jazyce. Umožňuje velmi jednoduše reprezentovat data formě primitivních datových typů, skládat z nich objekty, případně je organizovat v polích. Mezi hlavní výhody oproti XML patří vysoká úspora režijních znaků (CSV je ještě úspornější, ale nehodí se ke zpracování dat, která nejsou organizována v tabulce). XML naproti tomu dokáže lépe postihnout význam přenášených dat. JSON se dnes velmi často používá pro přenos v rámci intranetu, například je komunikačním formátem REST API.

Vstupní data

Webová služba bude číst data týkající se metrik vypovídajících o provozu služby nebo jejich jednotlivých součástí. Tato data budou uložena na straně poskytovatele a služba si je v případně

[10] Plné znění na http://tools.ietf.org/html/rfc4180
[11] Více na http://www.w3.org/XML/

zájmu stáhne ze specifikované adresy. Stažená data budou následně porovnána s případným definovaným prahem, na základě čehož pak bude rozhodnuto o plnění cílů metrik.

V rámci dohody o úrovni služeb je možné definovat dva základní typy metrik. První z nich jsou metriky vztažené k výkonnosti. Tyto jsou jednodušší – budou obsahovat pouze čas zachycení výkonnostní hodnoty a její hodnotu v daném čase. Hodnota (`captured_value`) musí být uvedena jako celé nebo desetinné číslo, v desítkové soustavě. Čas musí být uveden ve standardním `UNIX timestamp` formátu (počet vteřin od 1. 1. 1970). Zde je uvedena notace ve všech podporovaných formátech:

```
CSV:
captured_time; captured_value
captured_time; captured_value
…

XML:
<values>
      <value time="captured_time">captured_value</value>
      <value time="captured_time">captured_value</value>
      …
</values>

JSON:
[
      {
      time: captured_time,
      value: captured_value
      },
      {…},
      …
]
```

Druhým typem jsou metriky svázané s dostupností. Tyto fungují stejně jako výše zmíněně výkonnostní metriky – v případě potřeby si služba data sáhne. Liší se však ve významu. Zatímco výkonnostní metriky budou pouze porovnány s prahem, nad metrikami spojenými s dostupností bude vykonán výpočet. Vstupem totiž budou časy začátku výpadku a doba jeho trvání. Na základě těchto hodnot je pak možné provést výpočet kterékoli z metrik vztažených k výkonnosti uvedených v ITIL (služba bude pro jednoduchost počítat pouze celkovou ne/dostupnost služby za období, ostatní metriky by bylo velmi podobné, lišily by se pouze v provedené operaci). Čas výpadku (`failure_start_time`) bude uveden ve stejném formátu jako čas zachycení u výkonnostních metrik. Doba trvání výpadku (`failure_duration`) bude uvedena v sekundách.

```
CSV:
failure_start_time; failure_duration
failure_start_time; failure_duration
…

XML:
<values>
      <value time="failure_start_time"> failure_duration </value>
      <value time="failure_start_time"> failure_duration </value>
      …
</values>
```

```
JSON:
[
      {
      time: failure_start_time,
      value: failure_duration
      },
      {…},
      …
]
```

Pokud by tyto formáty nevyhovovaly danému použití, lze docela jednoduše definovat transformace, které je mezi sebou mohou libovolně převádět. K tomu můžou posloužit například XSLT transformace.

Výstupní data

Zde se jedná o formát dat, kterými bude služba kontaktovat posluchače (listeners), navázané na definovaný práh, pokud bude prolomen. Primárním účelem tohoto je možnost provázání služby s aplikacemi typu service desk. Více by tato služba dávala smysl, pokud by aplikace zpracovával nepřetržitý tok dat o aktuálních hodnotách metrik, informace by tak měla daleko vyšší hodnotu. Toto však bude ponecháno jako možné rozšíření, neboť se jedná o velmi netriviální záležitost. Ta by mohla být koncipována jako zvlášť vyčleněná webová služba, která by s touto kooperovala.

Výstupní data budou poskytována pouze v jednom formátu, neboť datový model je navržen tak, že neumožňuje, aby uživatel specifikoval výstupní formát. Rozhodl jsem se zvolit formát JSON, neboť tvoří pomyslnou zlatou střední cestu mezi příliš upovídaným XML a CSV, které je omezeno na tabulková data. Formát výstupních dat zasílaných na port posluchače je tedy následující:

```
{
      metric_id: id,
      threshold_id: id,
      thereshold: threshodl_value,
      values: [
            {time: captured_time, value: captured_value},…
      ]
}
```

Jedná se tedy o jednoduchý objekt zprávy, který identifikuje metriku, čas, kdy byl překročen práh, informace o získané hodnotě a stanoveném prahu.

4.3 Webový klient

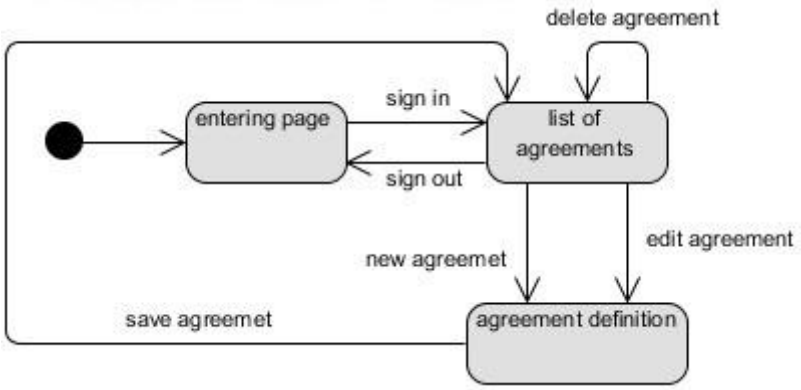
V této podkapitole je specifikován návrh ukázkové klientské aplikace. Postupně jsou rozebrány: návrh posloupnosti obrazovek, validace vstupů a v poslední podkapitole je krátce rozebrána bezpečnost. Některé požadavky nebo zde zmíněné návrhy nejsou implementovány. Toto nepovažuji za chybu, neboť celý tento klient je vypracován nad rámec zadání diplomového projektu. Snaha dokončit jej, je ale důležitá pro využití hlavní části, webové služby. Bez klientské aplikace by služba byla pouze komponentou. Ta by sice byla volně ke stažení, ale k praktickému využití by bylo nutné

buďto zapojit ji do fungujícího systému (toto umožňuje většina CMS a ERP systémů na trhu, namátkou mohu zmínit například SAP nebo produkt brněnské společnosti Kentico) nebo klientskou aplikaci vytvořit. Je možné, že některé části klientské aplikace bude třeba přizpůsobit nebo upravit, ale jádro aplikace je pro zamýšlený výukový účel dostačující.

4.3.1 Posloupnost obrazovek aplikace

Pro modelování posloupnosti obrazovek jsem využil diagram stavů. Na prvním diagramu (obrázek 4.8) je vidět pouze základní schéma. Uživatel přistoupí na přihlašovací stránku, vyplní zde přihlašovací údaje (login a heslo) a postupuje na rozcestník. Systém neumožňuje registraci uživatelů. To vychází z podstaty určení informačního systému –bude sloužit výhradně pro vnitřní účely fakulty a uživatelům takzvaně zvenčí nebude přístup za žádných okolností umožněn.

V případě úspěšné autentizace je uživatel přesměrován na rozcestník. Ten reprezentuje stránka se seznamem všech smluv, které uživatel vytvořil. Tyto smlouvy může upravit nebo smazat (je upozorněn na nenávratnost této akce). Další možností je vytvořit smlouvu novou. Obě možnosti (nová/editace) vedou na posloupnost stránek pracovně nazvaných definice smlouvy. Tyto jsou kvůli zvýšení přehlednosti uvedeny zvlášť v diagramu na obrázku 4.9.



Obrázek 4.8 Základní diagram obrazovek.

Po odhlášení je uživatel opět přesměrován opět na vstupní stránku. O poznání složitější je druhý diagram obrazovek. Ten reprezentuje stěžejní část aplikace – vytváření smlouvy. Tato posloupnost stránek by měla uživatele krok po kroku provést procesem tvorby dohody o úrovni služeb. Jednotlivé obrazovky jsou doplněny komentáři v souladu s terminologií rámce ITIL tak, aby uživateli co možná nejvíce tuto činnost ulehčily.

Po kliknutí na editaci smlouvy nebo vytvoření nové je uživatel přesměrován na stránku nazvanou „základní informace“. V diagramu je však naznačen vstup přes pseudo-stav „menu“. Taková stránka v rámci systému neexistuje. Jedná se o komponentu, která je vložen do všech stránek svázané s vytvářením dohody. Následující posloupnost stránek bude popsána tak, jak by na sebe měly teoreticky navazovat. Nic však uživatele nenutí se této posloupnosti držet, stejně tak jej nikdo nenutí

využít všechny specifikované možnosti. Tento princip je dle mého názoru plně v souladu s ideologií rámce ITIL – ten také nepřikazuje, pouze doporučí, dá uživateli na výběr.

Úvodní obrazovka tedy uživateli umožní zadat základní informace o dohodě, jako je její jméno, popis, popis pokrývané služby a další. V podstatě se jedná o všechny jednoduché atributy dohody, jak jsou specifikovány v konceptuálním diagramu u třídy `Agreement`. Odtud je doporučeno pokračovat specifikací smluvních stran. Ty jsou popsány svým názvem a rolí v daném kontraktu. Může jich být s dohodou svázán libovolný počet. Typicky se jedná o poskytovatele, zákazníka a blíže neurčený počet třetích stran. Ke každé straně je doporučeno uvést kontakt na osobu/oddělení odpovědné za podpis smlouvy. Rovněž doporučujeme doplnit i jména osob, které za každou smluvní stranu dohodu podepsaly spolu s datem podpisu. V případě, že uživatel při zadávání strany nebo podpisu udělá chybu, je možné danou položku odstranit a zadat znovu (toto samozřejmě platí i pro většinu dalších položek).

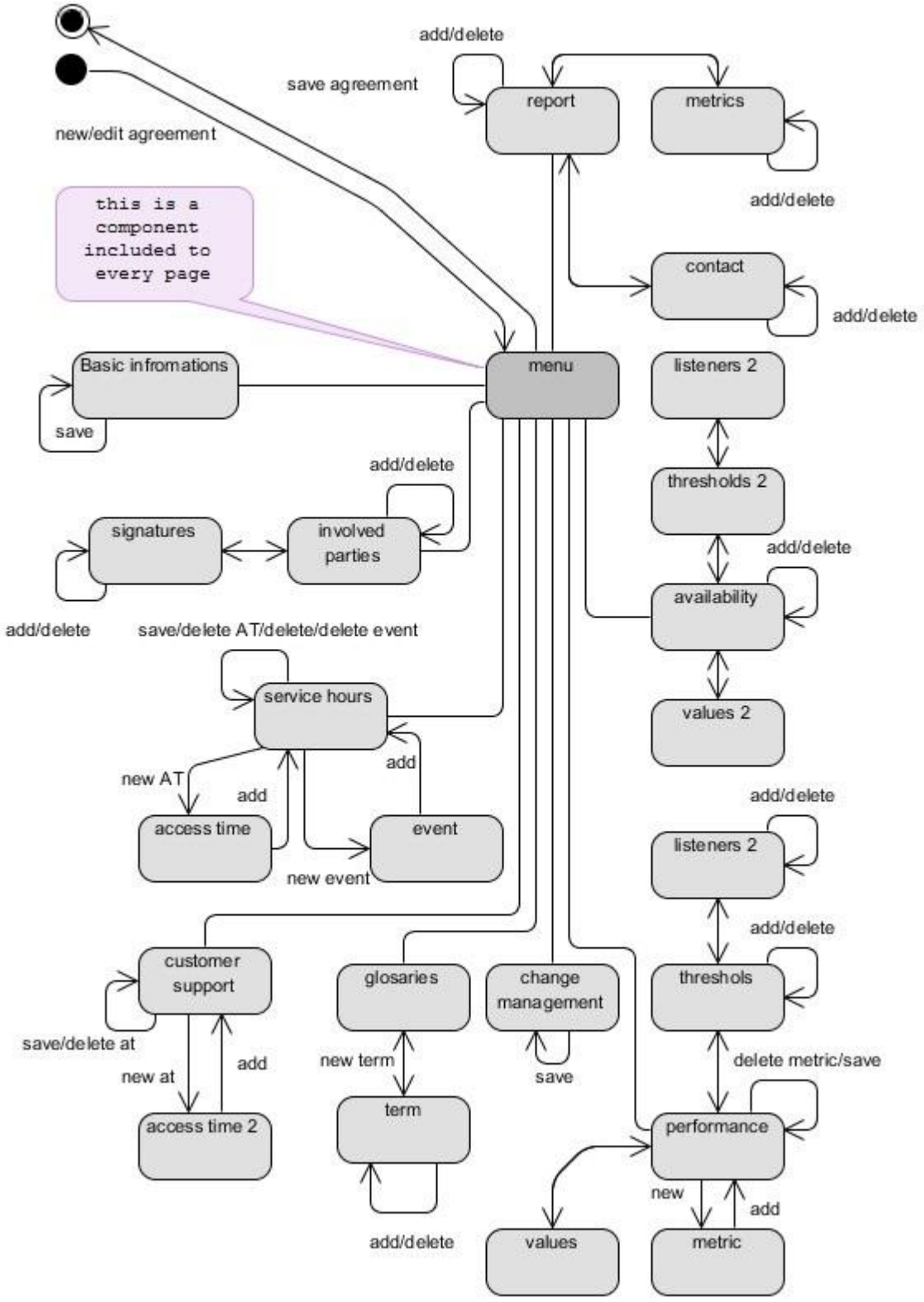
Po definici smluvních stran je vhodné specifikovat hodiny, kdy je možné službu využívat. Stěžejním bodem tohoto kroku je ovšem definovat přístupové časy a události se službou svázané. Při definici času dostupnosti je vždy vyžadován den a čas začátku a konce provozu služby. V případě provozu 24/7 bychom tedy definovali přístupový čas pro všech sedm dní vždy od času 00:00 do času 23:59. Události slouží k uložení jakýchkoli jednorázových akcí týkajících se služby. Patří sem například odstávka služby spojená s údržbou serverů a podobně. Událost je definována svým typem, popisem a datem začátku/konce.

S přístupovými časy pracuje uživatel i v případě definice zákaznické podpory. Kromě toho je umožněno uvést popis sloužící pro ujasnění podmínek, za jakých je zákazníkovi podpora dostupná, co všechno je/není její součástí. Velmi důležité jsou informace týkající se odezvy zákaznické podpory a to ne jenom určení maximálního času, do kterého musí uživatel odpovědět, ale i definice toho, co je za takovou odpověď považováno. Je možné uvést libovolnou poznámku – informaci, která se nehodí ani do jednoho z výše uvedených polí.

Další, tentokráte poměrně krátkou konfigurační položkou jsou informace týkající se managementu změn. Zde se jedná pouze o textové pole, ve kterém budou popsány podmínky, za kterých je možné zahájit změnové řízení, doplněné o kontakt na osobu/oddělení zodpovědné za změnové řízení na straně poskytovatele.

Stránka věnovaná slovníkům je poměrně jednoduchá. Je na ní možné definovat nový slovník, případně doplnit/smazat již existující. Do slovníku jsou zadávány dvojce term-definice. Definice termu může být obsáhlejší, term samotný je však holé slovo nebo slovní spojení. Jak již bylo zmíněno výše, slovníky slouží k ujasnění používaných pojmů a přednáší tak zbytečným nedorozuměním. Následuje skupina stránek věnovaných po řadě: výkonnosti, dostupnosti a reportům. Všechny tyto stránky jsou nějakým způsobem propojeny metrikami. I když jsou jim v návrhu věnovány pokaždé zvláštní stránky (z technických důvodů), budou popsány pouze jednou. Jako první bude probrána

obrazovka věnovaná výkonnosti. Zde se nachází pole věnované textovému popisu výkonnosti služby a odkaz pro definici metrik. Těch může být s výkonnosti svázáno několik, a tak je hned pod tímto



Obrázek 4.9: Diagram obrazovek - vytváření dohody.

odkazem nachází přehledová tabulka již specifikovaných metrik. Metrika je dána svým názvem a typem vstupu. Oba dva parametry jsou vyžadovány. Po definici se metrika objeví v tabulce. Zde je možné přejít na výčet naměřených hodnot nebo k definici prahových hodnot. Tím je uživatel přesměrován na novou stránku, kde může specifikovat práh (jeho hodnotu). Tyto prahy se zobrazují v tabulce, ze které je dostupný odkaz na definici odběratelů (posluchač). Ti jsou identifikováni adresou a portem.

Dostupnost se od výkonnosti odlišuje v několika ohledech. Prvním je fakt, že definicí dostupnosti může být na rozdíl od definice výkonnosti vícero. Dostupnost je však vždy svázána pouze s jednou metrikou. Na stránce se tedy definuje dostupnost současně s odpovídající metrikou. Obrazovky týkající se naměřených hodnot, definice prahů a posluchačů jsou stejné jako u definice výkonnosti.

Poslední zbývající obrazovkou jsou reporty neboli zprávy o provozu služby. U těch zadá uživatel vždy název, čas odeslání (hodina a den v týdnu/měsíci), frekvenci odesílání (denně, týdně nebo měsíčně) a krátký popis. Druhou částí je výběr metrik, které budou v reportu obsaženy. Uživateli je umožněno vybírat jak z metrik definovaných v rámci výkonnosti, tak z metrik definovaných v jednotlivých specifikacích dostupnosti služby. Reportů může být definován libovolný počet.

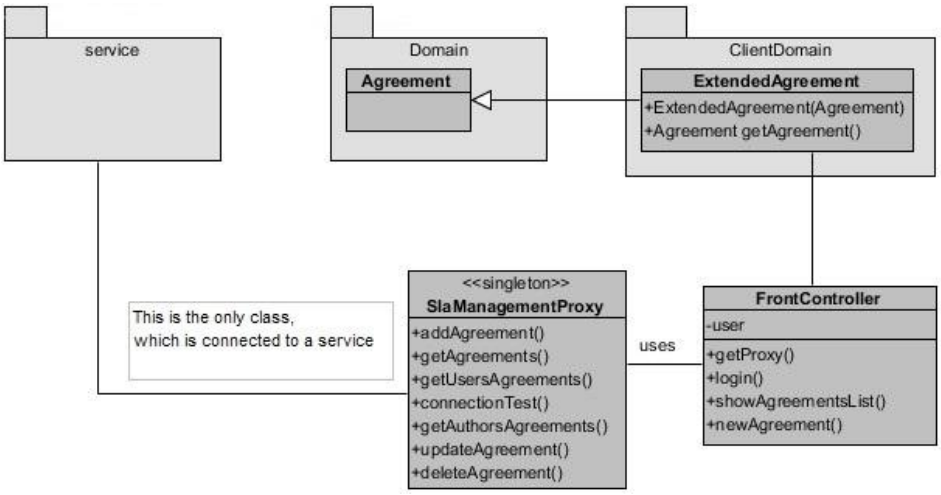
Z části aplikace věnované vytváření/úpravě dohody o úrovni služeb se uživatel navrací kliknutím na tlačítko uložit. V tomto případě je dohoda zahrnuta do seznamu již existujících a odeslána službě ke zpracování (v případě úpravy dohody je samozřejmě uložena úprava, nová dohoda se nevytváří).

4.3.2 Diagram tříd

Objektový klientské aplikace plně podřízen požadavku na využití architektury MVC[12]. Ta bude detailněji vysvětlena v části práce věnované implementaci. Návrh rovněž ukazuje zabezpečení přístupu ke službě a blíže ukazuje využití datového modelu (zde uvedeno jako doména).

Aplikační logika je v architektuře MVC obsažena v části nazvané Controller. V našem případě ji reprezentuje třída `FrontController`, která obsahuje pro každý zpracovávaný požadavek jednu metodu. Těch je poměrně velké množství, proto uvádíme v diagramu pouze ukázkové. Za zmínku stojí metoda `getProxy()`, která implementuje návrhový vzor `Singleton`. Ten zabezpečuje, aby se nepoužívala více než jedna instance třídy proxy v rámci kontroléru. Důležitý je rovněž atribut reprezentující uživatele. Tento je na straně serveru uchováván v rámci sezení a zajišťuje tak udržení kontextu.

[12] Model-View-Controller, softwarová architektura.



4.10: Zjednodušený diagram tříd klientské aplikace.

V rámci klientské aplikace musí být samozřejmě dostupné všechny třídy, které budou přenášeny mezi službou a klientskou aplikací. Tyto jsou obsaženy v balíčku `Domain` a bez výjimky se shodují se třídami, které jsou součástí stejnojmenného balíčku na straně služby (ten je uveden na obrázku 4.6 v předchozí kapitole). Problém tříd, které jsou přenášeny v rámci služby, je, že mohou obsahovat jenom omezený výběr datových typů. Webové sužby jsou totiž nezávislé na platformě, jinými slovy, všechny přenášené informace musí být možno vymodelovat v libovolném jazyce, který webové služby podporuje (ať už se jedná o Javu, C#, PHP, Python a další). Kromě primitivních datových typů můžeme použít[13] i řetězce a třídy reprezentující datum/čas. Jedinou možností jak pracovat s kolekcí prvků je využití pole.

Využití těchto datových typů není nijak limitující, neboť jazyk má i tak plnou výpočetní sílu. Práce s těmito třídami je však pro programátora nepohodlná a náchylná k chybám. Proto je vytvořen balíček `ClientDomain`. V tom jsou obsaženy všechny třídy, které dědí od základních doménových tříd. Předpokládané využití je primárně u operací s poli. Je však nutné, aby každá z těchto rozšiřujících tříd obsahovala konstruktor, který bude mít jako parametr jednoduchou variantu této třídy. Další podmínkou je metoda, která umožní z rozšířené třídy získat její jednoduchou variantu. V diagramu na obrázku 4.10 je toto naznačeno ve třídě `ExtendedAgreement`, která má konstruktor s parametrem třídy `Agreement`. Stejně tak obsahuje metodu `getAgreement()`, která vrací jednoduchou variantu sebe sama (tedy třídu `Agreement`).

Pro komunikaci s webovou službou, tedy pro odesílání požadavků a zpracování odpovědí slouží třídy obsažené na stran klienta v balíčku `Service`. Z pohledu kontroléru je nejdůležitější třída `SlaManagementProxy`. Ta již byla zmíněna. Obsahuje stejné metody, které veřejně poskytuje služba. Jejich volání pak systém zpracuje a přetransformuje na volání metod vlastní služby.

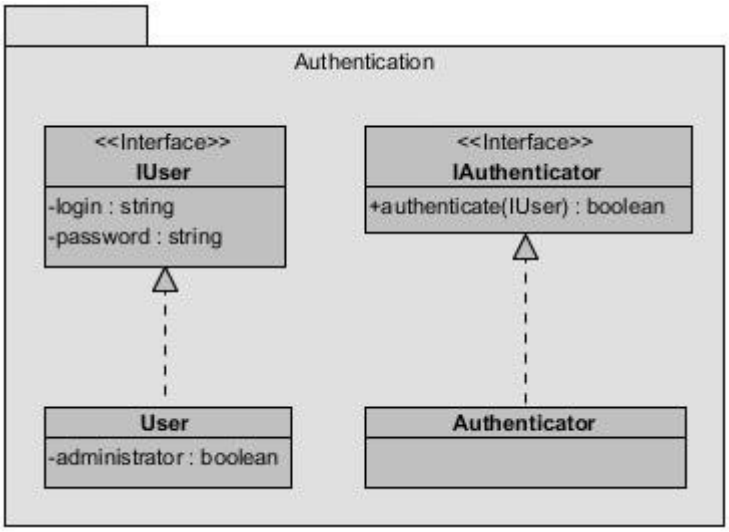
[13] Úplný přehled povolených datových typů je uveden například zde:
http://docs.oracle.com/javaee/6/tutorial/doc/bnazc.html

Programátor tedy pracuje se třídou stejně, jako by využíval metodu běžné lokální třídy a obsažené volání webové služby je mu skryto.

4.3.3 Autentizace

Jak již bylo zmíněno v požadavcích, klientská aplikace nebude nijak řešit správu uživatelů. Přesto je ale nutné nějakým způsobem kontrolovat přístup uživatelů. Bohužel není možné vytvořit řešení takříkajíc na míru fakultě, tedy specializovanou autentizaci vůči LDAP serveru, neboť jako studenti nemají k fakultnímu LDAP serveru přístup, z bezpečnostních důvodů. Další důvodem by byla ztráta obecné uplatnitelnosti aplikace, v neposlední řadě hraji roli i časové hledisko.

Návrh je tedy veden tak, aby mohl být s možná nejnižším úsilím dosazen jakýkoli autentizací mechanizmus – ať už LDAP server, nebo třeba zvláštní databáze a tak podobně. Tohoto docílím navržením dvou jednoduchých rozhraní. Jak je možné vidět na obrázku 4.11.



4.11: Baliček sloužící s rozhraními určenými pro autentizaci.

Rozhraní `IUser` vymezuje dva atributy bezpodmínečně nutné pro přihlášení, a sice přihlašovací řetězec a heslo. Oba dva parametry jsou definovány jako typ řetězec, důvodem je snadná možnost přetypování pro případ potřeby. Druhé rozhraní reprezentuje třídu, která uživatele autentizuje. Jak to udělá (vůči čemu a jak bude uživatel autentizován) je čistě starost třídy. Jedinou metodou deklarovanou v rozhraní je metoda `authenticate(IUser)`, která vrací hodnotu `true` v případě úspěšné autentizace.

4.3.4 Validace

Jako u většiny informačních systémů je nutné kontrolovat, jaká data uživatel zadává. Toto má hned několik důvodů. Prvním z nich je snížení bezpečnostního rizika, že uživatel zadá nebezpečný kód. Této problematice je věnována následující kapitola. Druhým důvodem je usnadnění práce uživateli,

který se nesnaží aplikaci nijak zneužít, ale naopak ji využívá pro práci, je vždy pozitivní, když je upozorněn na možnou chybu. Validaci vstupních dat u webových projektů je možné provádět dvěma základními způsoby – na straně klienta a na straně serveru. Obojí má své výhody i své nevýhody. Oba způsoby se ale skvěle doplňují.

První možností je validovat data již na straně tenkého klienta, tedy ve webovém prohlížeči. Toto je poměrně elegantní a rychlé řešení. Zadaná data se totiž nemusí odesílat na server, který je validuje a následně v případě neúspěchu posílá odpověď, ale jsou zpracovávána na stroji uživatele. Většinou jsou data validována prostřednictvím uživatelského JavaScriptu. Ten, i když je v dnešní době dostupný na většině počítačů i mobilních zařízení, je hlavní slabinou validace na straně klienta. Jsou totiž stále uživatelé, jejichž prohlížeč buďto JavaScript nepodporuje, nebo jej v prohlížeči schválně zakázali (toto často zdůvodňují zbytečným zatížením prohlížeče nebo bezpečnostní hrozbou).

Oproti tomu validace na serveru je prováděna až po odeslání vyplněných dat, jak již bylo zmíněno. Hlavní výhodou je, že tato validace proběhne vždy, jelikož všechny odeslané požadavky jsou zpracovávány na straně serveru. Nevýhodou je možná časová prodleva, než se uživatel dozví, že zadal nevalidní data. Vhodným řešením tedy je provádět validaci jak na straně klienta, tak na straně serveru. U většiny uživatelů jsou data validována okamžitě při zadávání a na sever jsou odeslána ve správné formě. Uživatel tak nedostane zamítavou odpověď. V případě, že uživatel nebude mít dostupný JavaScript, může na server odeslat nevalidní data, ta ale nebudou dále zpracována, neboť proběhne jejich druhá validace a uživatel je na chybu upozorněn.

Se standardem HTML5 přichází nový atribut `type` formulářového prvku `input`, který umožňují specifikovat formát/type vyžadovaných data. V takovém případě jsou data validována přímo prohlížečem, není potřeba doplňovat žádný JavaScript. Prohlížeč, nejenom, že uživatele upozorní na chybu v jazyce prohlížeče, ale vstupů vázaných například s datem nebo časem vykreslí i jednoduché a vkusné uživatelské rozhraní, které napomůže rychlejšímu zadání dat ve správném formátu. Tyto prvky uživatelského rozhraní bylo dříve nutné doplňovat pomocí JavaScriptových knihoven. Slabou stránkou je prozatímní neúplné podpora těchto prvků napříč všemi prohlížeči. Většina prohlížečů zatím implementuje pouze vybrané typy. Úplný přehled typů vstupních dat podle HTML5 je dostupný na webu[14] konsorcia W3C. U každého typu je doplněn i výčet prohlížečů, které daný prvek podporují.

4.3.5 Bezpečnost

Tato část práce krátce zmiňuje bezpečnostní požadavky kladené na webový informační systém. Oba dva jmenované typy útoků jsou velmi vysokým rizikem a je bezpodmínečně nutné, aby byly brány v úvahu při návrhu každého webového informačního systému. U každého ze dvou typů útoků je vždy

[14] http://www.w3schools.com/html/html5_form_input_types.asp

krátce uveden teoretický základ a následuje navrhované protiopatření, které by mělo tomuto typu útoku zabránit.

Cross-Site Scripting (XSS)

Problematika takzvaných XSS útoků je natolik rozsáhlá, že jsou jí věnovány celé knihy, proto bude uveden pouze krátký úvod. Celá tato kapitola vychází z informací uvedených v [10]. Jedná se o jeden z nejrozšířenějších útoků v prostředí webu. Útočník se snaží nějakým způsobem do stránky vložit vlastní skript, který je následně po vložení do stránky zpracován na straně klienta a pomocí něho napadnout uživatele. Nebezpečí spočívá v tom, že uživatel tato závadná data obdrží z důvěryhodného zdroje. Možná škoda způsobená tímto útokem je pak úměrná cennosti dat, která takto může útočník odcizit

Útoky je možné členit podle několika hledisek. První z nich je dělení podle trvanlivosti. *Perzistentní útoky* jsou takové, kdy útočník dokáže vložit do stránky škodlivý kód trvale, například tím, že jej vloží do databáze, třeba jako komentář a stránka jej pak sama zobrazuje. Takto může jednoduše ukrást obsah cookies používaných pro identifikaci přihlášeného uživatele. Napadený uživatel pak nemusí klikat na žádný upravený odkaz. Při *neperzistentním útoku* není do stránky vkládán škodlivý kód trvale, ale využívá se možnosti přenášet obsah proměnných metodou GET přímo v adrese. Pokud je obsah této proměnné následně na stránce zobrazen bez jakéhokoli ošetření (například pomocí PHP skriptu), tak stačí, aby útočník poslal uživateli email s odkazem směřujícím na důvěryhodnou stránku, ve kterém bud místo očekávaného obsahu proměnné zakódován škodlivý kód. Tento se pak zobrazí a provede na straně klienta. Dalším často zmiňovaným typem útoku je takzvaný *DOM útok*. Tento oproti dvěma zmíněným nevyužívá změny kódu stránky přímo na serveru, ale provede změnu až na straně klienta a sice tak, že některé stránky využívají JavaScriptový kód pro zobrazení části URL (stejně tak jako u uvedeného neperzistentního útoku může být například zobrazena hodnota proměnné v metodě GET). Útočníkovi tak k provedení útoku opět stačí uživateli zaslat vhodně pozměněný odkaz směřující na napadenou stránku.

Abychom zabránili těmto útokům, je důležité dodržet jednoduché bezpečnostní zásady. Proti prvním dvěma typům útoků se můžeme bránit důslednou úpravou (odstranění HTML značek escapováním) všech zobrazovaných hodnot na straně serveru. Poslednímu uvedenému typu útoku zabráníme stejnou metodou, avšak na straně klienta.

SQL Injection

Dalším častým typem útoku je injekce SQL kódu. Jeho popisu a prevenci se věnují desítky článků, jako například [11], kde je uveden hezký přehled a ze kterého tato kapitola vychází. Jedná se typ útoku založený na vsunutí vlastního kódu do SQL kódu aplikace – tedy její databázové vrstvy. Využívá bezpečnostní mezery v aplikaci způsobené nesprávným vkládáním vstupních dat do

předpřipraveného SQL příkazu. Tento útok se dělí na několik různých typů. Zde jsou uvedeny pouze základní. Detailnější přehled je uveden v [11].

Prvním a základním typem útoku je využití neodfiltrování escapovacích znaků. Pak stačí, abychom například kód pro přihlášení do aplikace doplnili o `OR 1' = '1'; --` . Příkaz nejenom, že bude vždy vyhodnocen jako `TRUE`, ale případná zbývající část dotazu SQL dotazu bude považována za komentář, tedy se neprovede. Druhou možností je využití špatné kontroly typů, ta hrozí u slabě typovaných jazyků. Pokud programátor například očekává jako vstupní parametr celé číslo, ale důsledně nezkontroluje, zda předaná data jsou tohoto typu, stačí, aby útočník připojil za celé číslo například kód `=1; DROP TABLE users ; --` a příkaz odstraní celou tabulku uživatelů.

Zvláštní kategorií útoků jsou takzvané Blind SQL Injection. Tyto jsou charakteristické tím, že i když je webová stránka k tomuto typ útoku náchylná, výsledky útoku nejsou útočníkovi přímo viditelné (stránka nezobrazí přímo výsledky útoku, ale zobrazuje se rozdílně v závislosti na ne/úspěchu útoku). Útočník postupuje tak, že vyzkouší, jak stránka reaguje na útok, v případě, že je SQL příkaz splněn, nebo nesplněn. Na základě toho pak může zjišťovat informace, jako je verze serveru a podobně.

Existuje několik možných opatření pro snížení pravděpodobnosti úspěšného útoku. Na straně aplikace je základním opatřením používání předpřipravených dotazů (prepared statements). Kód dotazu je připraven dopředu, hodnoty jsou nahrazeny proměnnými a při volání jsou pouze speciální funkcí/metodou doplněny hodnoty parametrů. Takto připravené dotazy jsou nejenom bezpečnější, ale i rychlejší. Rovněž použití knihoven pro objektově relační mapování zabraňuje injekci SQL kódu, neboť programátor přímo nevytváří SQL kód pro DBMS.

Na straně databázového serveru je doporučené vhodně nastavit přístupová práva danému uživateli. V praxi to znamená vytvořit 2databázové uživatele. Jeden z nich bude mít vyšší práva a bude sloužit pro administraci databáze provozovatelem. Druhý z nich je vytvořen pro přihlašování webové aplikace k databázi a má práva omezená na nutné minimum (většina webových aplikací si vystačí s právy k příkazům `SELECT`, `INSERT`, `UPDATE` a `DELETE`).

Webový klient sám o sobě nebššude v současné verzi přistupovat k databázi, bude pouze komunikovat s webovou službou, která veškerou činnost spojenou s databází obstará. Webová služba bude používat framework pro OR mapování, tím se dostatečně sníží riziko napadení SQLI útokem. Možným rizikem by mohlo být případné neopatrné rozšíření aplikace.

5 Implementace

V této kapitole je rozebrána implementační část práce. Webové službě i klientské aplikaci je věnována celá kapitola popisující volbu využívaných technologií, jejich výhody a nevýhody, případně krátké porovnání s jinými dostupnými technologiemi, které se nabízely k využití.

5.1 Webová služba

Již z principu webových služeb vyplývá, že jsou nezávislé na programovacím jazyce, který je implementuje. Rozhodujícím faktorem je dodržení komunikačního protokolu, který je součástí standardu webových služeb. Tato kapitola postupně vysvětluje volbu programovacího jazyka, implementaci datového modelu a v závěru obsahuje zmínku o testování.

5.1.1 Programovací jazyk

Jak bylo zmíněno, webové služby mohou být implementovány v libovolném programovacím jazyce. S ohledem na druhou část práce, kterou je webový klient, jsem ale volil pouze z platforem určených pro tvorbu webových aplikací. Druhou nutnou podmínkou bylo, aby zvolené prostředí obsahovalo potřebné knihovny pro obsluhu webových služeb. Tyto jsou standardní, časem a většinou mnoha miliony uživateli prověřené. Vytvoření vlastních knihoven pro tuto činnost by nebylo nikterak přínosné.

PHP

Prvním zvažovaným jazykem bylo PHP (PHP: Hypertext Preprocessor, dříve Personal Home Page). Jedná se o skriptovací programovací jazyk fungující na straně serveru, nezávisle na platformě. Autorem původní verze je Rasmus Lerdorf, který ji naprogramoval pro vlastní účely ve formě několika skriptů v jazyce Perl. Když zjistil, že je o ně zájem, publikoval je pod názvem Personal Home Page Tools. Později doplnil skripty pro analýzu vstupů z HTML formulářů a tím položil základ první použitelné verzi PHP. To se stalo velmi oblíbené a vývoj přešel z jednoho člověka na skupinu vývojářů. Ta opět rozšířila jádro jazyka tak, aby mohli i ostatní vývojáři volně přidávat nové funkce. Později byl jazyk přepsán z Perlu do C/C++, rozšířen o mnoho knihoven a doplněn o možnost programovat objektově orientovaným způsobem [12]. V současné době se nachází ve verzi 5.5.12. Jazyk obsahuje balíky objektu určených pro práci s webovými službami. Jejich kompletní popis se dá nalézt na webu věnovaném jazyku PHP[15].

Nevýhodou jazyka je, že není nijak kompilován. Skript je tedy při každém požadavku na server nutné zavést do paměti, přeložit jej do vnitřního kódu a ten teprve interpretovat. To s sebou nese

[15] http://php.net/manual/en/refs.webservice.php

nejen určité zdržení v případě rozsáhlejších projektů, ale rovněž to zamezuje možnosti výkonnostních optimalizací, které je u kompilovaných jazyků možno provádět (pokud je program běží delší dobu a pokaždé jsou mu jenom předhozeny jiné hodnoty svázané s daným požadavkem, může běhové prostředí vytvářet statistiky běhu programu a na jejich základě pak jeho běh optimalizovat). Hlavním argumentem mluvícím pro použití PHP při programování práce je jeho vynikající znalost ze strany autora. Jelikož je ale jazyk nekompilovaný, nebyl nakonec použit.

ASP .NET

Technologie ASP.NET společnosti Microsoft je nástupcem klasického ASP. To sloužilo jako klasický skriptovací jazyk. Hlavní výhodou začlenění ASP do platformy .NET je kompilace tzv. *behind code* (tedy obslužného kódu stránky) do mezi jazyka Common Intermediate language(CIL), který pak běží na virtuálním stroji (Common Language Runtime – CLR). Běh programu je tedy mnohem rychlejší než u běžného skriptovacího jazyka. Druhou výhodou, která pramení ze začlenění do .NET platformy je množnost volby jazyka, ve kterém bude *blind code* napsán. Do CIL je totiž možné přeložit C#, VisualBasic, Deplhi, F#, J# nebo například C++. Mezi další výhody patří možnost znovupoužít šablon (vedoucí k redukci duplicity kódu), velká škála dostupných knihoven a podobnost s vývojem desktopových aplikací pro Windows, což umožňuje jednodušší přechod od jednoho prostředí ke druhému. Asi nejpoužívanější technologií pro vývoj webových aplikací v prostředí .NET jsou Webforms. Zde platí, že každá vytvořená stránka může obsahovat pouze jeden formulář a je pevně svázána s *blind code*, který ji obsluhuje. Od roku 2008 vyvíjí Microsoft novou platformu pro tvorbu webových aplikací nad .NET – MVC framework. Podpora WebForms ale nekončí. Pro tvorbu a využití webových služeb nabízí ASP.NET standardní podporu[16], která je jako naprostá většina knihoven v prostředí .NET velmi dobře zdokumentovaná.

Java EE

Platformou, ve které byla nakonec webová služba i její klientská aplikace vyvinuta se stala Java EE (*Enterprise Edition*). Tato technologie je součástí platformy Java a je primárně určena pro vývoj informačních podnikových systémů. Je postavena na základech Java SE (*Standard Edition*), ve které jsou vytvořeny jednotlivé komponenty tvořící EE platformu. Její vznik se datuje zhruba na přelom tisíciletí, kdy byla vydána verze 1.3. V současné době je aktuální verze Java EE 7, vydaná v roce 2013.

Java EE umožňuje programátorům vytvářet poměrně jednoduše rozsáhlé distribuované systémy díky svému několikavrstvému modelu. První z nich, na straně uživatele, je tenký klient. Následuje Java EE server, na kterém jsou ve webové vrstvě zpracovávány JSP stránky a business vrstvě pak Enteprise Java Beans. Poslední zmiňovanou vrstvou je databázový server [13]. JSP (JavaServer Pages) je jazyk určený pro tvorbu dynamických webových stránek. Vyvinula jej

[16] http://msdn.microsoft.com/en-us/library/aa286485.aspx

společnost Sun Microsystems, v současné době dceřiná firma společnosti Oracle. Vlastní psaní kódu v JSP se velmi podobá PHP nebo čistému ASP. V podstatě jde o zápis Java kódu uvozeného speciálními značkami do HTML kódu. Tento kód je přeložen do podoby servletu a spouštěn na serveru, například Apache Tomcat.

Java EE umožňuje již od verze 5 pomocí svého XML API vytvářet jak webové služby, tak jejich klientské aplikace na velmi vysoké úrovni. Není třeba řešit žádné nízko-úrovňové programování. V prostředí je tedy dostupná knihovna JAX-WS[17] (Java API for XML Web Services), která umožňuje vývojářům psát jak služby orientované na zprávu, tak RPC (Remote-Procedure Call) orientované služby. Dík tomuto můžou být podnikové aplikace psané v prostředí Java EE napojeny na komponenty napsané v libovolném jiném jazyce, což jim dodává daleko větší potenciál.

5.1.2 Objektově-relační mapování

Většina aplikací se snaží nějakým způsobem abstrahovat realitu, o které chce uchovávat určité informace. K tomu se používají programovací nejrůznější techniky. Objektově orientované programování patří v současné době mezi nejrozšířenější z nich. Datový model aplikace je zde reprezentován sadou různých objektů reprezentujících svůj reálný obraz a vztahy mezi nimi. Problém však nastává ve chvíli, kdy je potřeba tato perzistentní data uložit i mimo běh aplikace (řeší otázku, co s těmito daty po konci běhu aplikace). I když existuje řada objektově orientovaných databází, stále jsou velmi rozšířené databáze relační. Tyto dovedou většinou ukládat pouze data poskládaná z jednoduchých datových typů. Pro uležení objektových dat je tedy nutné definovat, jak a kam se mají které objekty mapovat v relační databázi. Úkolem ORM knihoven je tedy poskytovat programátorovi takovou vrstvu mezi databází a objektově orientovaným jazykem tak, aby mohl intuitivně pracovat přímo s instancemi tříd, jejichž veškeré změny budou persistentně uchovány v databázi. Programátor pak nemusí psát čisté SQL dotazy, aby získal data a mapoval je na objekty, to za něj obstará daná knihovna (často však poskytují objektového dotazovacího jazyk, OQL, jakési obdoby SQL pro kladení dotazů nad objekty). Vedlejším produktem ORM knihoven je odstínění od konkrétního databázového systému, neboť tyto knihovny často poskytují možnost nasazení na různé databáze.

Hibernate

Jedním z nejrozšířenějších frameworků pro OR mapováni používaných v Javě je Hibernate. Začátek jeho vývoje se datuje do roku 2001, kdy jej vytvořil Gavin King. Později byl odkoupen společností JBoss, kterou nyní vlastní Red Hat. Hibernate umožňuje dva způsoby, jak mapovat objekty na relační databázi. Prvním z nich jsou anotace, které jsou doplněny buďto přímo atributům objektů nebo jejich getterům. Druhou možností je pak využití mapovacích souborů, (Hibernate mapping files, `*.hbm.xml`) – metadat doplněných ke každému objektu, u kterého vyžadujeme perzistenci [14].

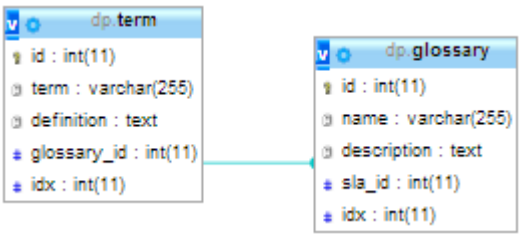
[17] http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html

Jedná se XML soubory s danou syntaxí. Perzistentními objekty nemohou být Java Beans, ale pouze takzvané Plain Old Java Objects (často se používá zkratka POJO). Takové objekty by neměly být nijak závislé na jiných (dědit od nich, implementovat rozhraní) a navíc pro všechny atributy musí existovat gettery a settery (metody pro získání a nastavení hodnoty privátního parametru, český ekvivalent není). Ke každému z těchto objektů je pak doplněn odpovídající mapovací soubor. Viz objekt **`Glossary`** v následující ukázce.

Glossary.java

```
public class Glossary {

	private int id;
	private String name;
	private String description;
	private Term[] terms;

	public Glossary(){
	}

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getDescription() {
		return description;
	}
	public void setDescription(String descritpion) {
		this.description = descritpion;
	}

	public Term[] getTerms() {
		return terms;
	}
	public void setTerms(Term[] terms) {
		this.terms = terms;
	}
}
```

K objektu `Glossary` patří mapovací soubor `Glossary.hbm.xml`, který jej mapuje na relační MySQL tabulku `glossary`. Její strukturu společně s tabulkou Term můžete vidět na obrázku 5.1.



5.1: Datový model v relační databázi MySQL.

Objekt třídy `Glossary` obsahuje pole termínů, čímž vytváří slovník. Pole bylo využito, aby bylo možné celý datový model komunikovat prostřednictvím webové služby (platí zde již zmíněné omezení na jazykově nezávislé datové typy, jinak by bylo možné využít vyšších datových typů, například kolekcí. Stejně tak Objekty třídy `Glossary` jsou prvky pole. V obou tabulkách a mapovacích souborech je tedy obsažen atribut `idx`, který reprezentuje pořadí v tomto poli).

Glossary.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 10.3.2014 20:09:06 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="xdrozd07.dt.domain.glossary.Glossary" table="glossary">
        <id name="id" type="int">
            <column name="id" />
            <generator class="identity"/>
        </id>
        <property name="name" type="java.lang.String">
            <column name="name" />
        </property>
        <property name="description" type="java.lang.String">
            <column name="description" />
        </property>
        <array name="terms" cascade="all">
         <key column="glossary_id"/>
         <list-index column="idx"/>
         <one-to-many class="xdrozd07.dt.domain.glossary.Term"/>
        </array>
    </class>
</hibernate-mapping>
```

Další nespornou výhodou použití hibernate je, že funguje jako určitá abstraktní vrstva mezi aplikačním kódem a databází. Pro připojení k databázi využívá takzvaných konektorů, které jsou závislé na konkrétním typu databázového systému. Pokud se tady programátor rozhodne z nějakého důvodu změnit databázovou technologii, tak by mu z aplikačního hlediska mělo stačit pouze vyměnit daný konektor (věc se pak samozřejmě komplikuje v případě nutnosti transformovat data z jedné databáze do druhé, ale i pro to existují automatizované nástroje, takže pokud programátor nevyužívá při definici databáze přemíru technologicky specifických definicí, i toto se dá zvládnout).

5.1.3 Databáze

Při výběru vhodné databáze pro tento projekt padla volba na databázový systém MySQL a to primárně z důvodu zkušeností autora s tímto systémem, dobrou podporu systému, jednoduchou instalaci a dobrou dostupnost i na naší fakultě. Svou roli hrála i dostupnost systému pro nekomerční použití pod licencí GPL.

MySQL je databázový systém vyvíjený společností Sun Microsystems, kterou v současné době vlastní Oracle. Jedná se svou velikostí o méně rozsáhlý databázový systém, který se již od svého prvopočátku soustředí primárně na rychlost. Na druhou stranu je to mírně na úkor jiných funkcí,

například složitějších možností zálohování. Architektura umožňuje doplňovat systém o různé enginy, některé lze připojovat i za běhu formou pluginů, což dává administrátorovi systému velmi rozsáhlé možnosti konfigurace [15]. V projektu byl využit engine InnoDB, který oproti hojně rozšířenému MyISAM provádí kontrolu cizích klíčů v definovaných tabulkách.

Transakce

Stejně jako většina jiných relačních databázových systémů poskytuje možnost využití transakcí. Jedná se o posloupnost SQL příkazů vztahujících se k jedné konkrétní operaci (například převod částky z účtu na účet, kde je třeba nejdříve snížit stav jednoho účtu a následně o stejnou částku zvýšit stav účtu jiného). Systém řízení báze dat (SŘBD) pak zaručí, že pokud při zpracování série dotazů dojde k libovolné chybě, navrátí se databáze do stavu před započetím dotazu. Jinými slovy se příkazy buďto provedou všechny korektně, nebo se neprovede žádný (pokud by ve zmíněném příkladu došlo k chybě při navyšování stavu účtu příjemce, systém neprovede transakci jako celek a odesílatel tak nepřijde o své finance, jen bude upozorněn, že se daná operace nezdařila). Databázovou transakci charakterizuje čtveřice vlastností:

- *Atomicita* – musí fungovat jako jednotka práce, nedá se štěpit,
- *konzistence* – databáze musí vždy přejít z jednoho konzistentního stavu do druhého,
- *izolace* – dokud není daná transakce provedená, jsou její výsledky pro ostatní transakce neviditelné,
- *stálost* – jakmile jednou systém potvrdí výsledek transakce, jsou změny stálé, musí být zachovány i pro případ havárie systému.

Tyto vlastnosti se často označují zkratkou z první písmen jejich anglických názvů jako ACID. Zaručení těchto vlastností vyžaduje vyšší výkon serveru (paměť, CPU i harddisk). Díky architektuře MySQL je však možné tuto vlastnost vypnout, pokud není potřeba a je tedy pouze na administrátorovi, aby databázi nastavil na míru dané aplikaci [15].

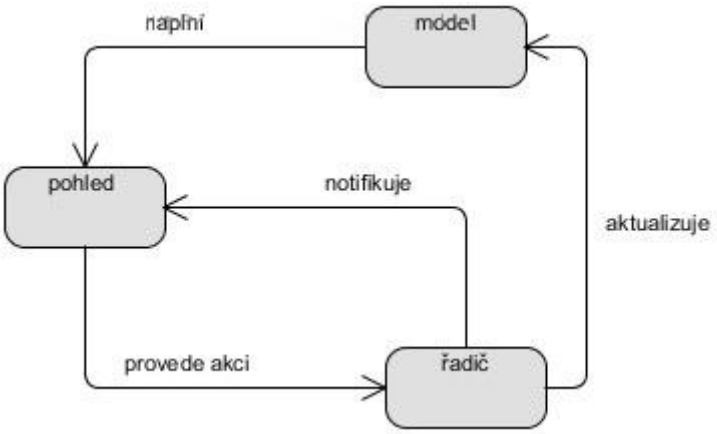
5.2 Webový klient

Klientská aplikace byla implementována jako klasická webová aplikace s tím, rozdílem, že vůbec nevyužívá databázi. Datovou vrstvu totiž reprezentuje webová služba vytvořená v předcházející části práce. Pro lepší udržitelnost kódu byl využit Framework Spring, který využívá architekturu MVC. Jak architektura tak Spring jsou v této kapitole krátce popsány. Kromě nich byly samozřejmě použity základní technologie nutné pro tvorbu webových aplikací, jako je HTML, kaskádové styly pro definici vzhledu stránky (vzhled použitý u klientské aplikace vychází z volně použitelné šablony k ukázkovému příkladu PHP frameworku Nette, který byl vhodně doplněn pro účel aplikace).

Další použitou technologií, která stojí za zmínku je JavaScript, který byl použit pro validaci vstupů z formulářů na straně klienta. K tomu výborně posloužil zásuvný modul volně dostupné knihovny jQuery[18].

5.2.1 Architektura Model – View Controller

Tato softwarová architektura, zkráceně označována jako MVC, byla poprvé použita v jazyce Smalltalk již v osmdesátých letech minulého století. Klade si za cíl rozdělit aplikaci na tři základní vrstvy taky, aby jakákoli úprava jedné z nich co nejméně ovlivnila zbývající dvě. Někdy je označována za návrhový vzor, ale většina autorů se spíš kloní k označení softwarová architektura, neboť řeší komplexnější problém [9].



5.2: Schéma modelu MVC.

Tyto tři vrstvy jsou pohled (`view`), řadič (`controller`) a model. Zjednodušeně by se dala jeho činnost popsat následovně: uživatele provede nějakou akci v pohlednu. Ten upozorní řadič, který akci zpracuje. V rámci tohoto naplní pohled daty z modelu a aktualizuje pohled.

Tento model je velmi oblíbený u vývojářů platforem pro tvorbu webových aplikaci. Implementují ho například PHP frameworky Nette a Zend, Ruby on Rails pro jazyk Ruby nebo již zmiňovaná platforma ASP .NET MVC od společnosti Microsoft.

5.2.2 Aplikační rámec Spring

Při vývoji webové aplikace byl využit opensource aplikační rámce Spring. Tento byl vytvořen a poprvé publikován v roce 2003. Je napsán v programovacím jazyce Java, pod licencí Apache licence. Umožňuje jednoduchý a velmi rychlý vývoj enterprice aplikací. Jedná se o modulární rámec, vývojář si tedy může vybrat ze široké škály knihoven a využít je, pokud to uzná za vhodné. Při práci byly použity například moduly AOP, web, JDBC, Context a samozřejmě Core – jádro frameworku.

[18] jQuery Validation dostupné z http://jqueryvalidation.org/

6 Závěr

Tato zpráva je věnována popisu analýzy, návrhu a vývoje programové části diplomové práce zaměřené na tvorbu webové služby pro správu SLA dohod. Ta byla implementována v prostředí Java EE společně s ukázkovým webovým klientem. Obě dvě části budou použity při výuce v předmětu INI na naší fakultě a jsou publikovány na serveru GitHub[19].

První část je věnována více teorii spojené s rámcem ITIL, druhá polovina je věnována návrhu a implementaci aplikace. Jako první byl v krátkosti představen rámec pro poskytování služeb v IT Infrastructure Library – ITIL, bylo zmíněno několik zásadních definic. Následuje stručná historie rámce a popis aktuální verze. Dále je představen životní cyklus služby podle ITIL.

Třetí kapitola je věnována specifikaci požadavků. V první polovině je doplněno několik nezbytných definic, ze kterých se dále vychází při specifikaci funkčních a nefunkčních požadavků na výslednou službu a klientskou aplikaci. Jako zásadní jsou vyhodnoceny požadavky na strukturu dohody. Ta je detailně rozepsána na začátku čtvrté kapitoly věnované návrhu služby. Popis dohody je odvozen od specifikace dohody v rámci knihy věnované návrhu služeb podle ITIL. Na základě tohoto popisu byl prezentován konceptuální model dohody, tedy jakýsi hrubý návod, jak uchovávat dohodu ve formě strukturovaného dokumentu.

Dále je v kapitole věnované návrhu prezentováno několik diagramů zachycujících jednotlivé vrstvy navrhované aplikace. Po nich následuje návrh webového klienta, zvláštní pozornost je věnována posloupnosti obrazovek doplněných vždy o patřičný komentář tak, aby na základě toho mohla být aplikace rychle implementována. Krátká poznámka je věnována bezpečnosti a validaci dat.

Pátá kapitola představuje technologie, které byly využity při tvorbě aplikace, jejich přehled, možné konkurenty a důvod volby té které konkrétní technologie.

6.1 Možná rozšíření

Při vývoji bylo identifikováno několik možných rozšíření aplikace. Jejich implementace by byla však natolik časově náročná, že nebylo možné je do aplikace zapracovat. První z nich se týká zpracování vstupních hodnot metrik týkajících se výkonnosti nebo dostupnosti služby. Aplikace očekává konečné vstupní hodnoty a nijak dál je netransformuje. Pro reálné využití v komerčním prostředí by bylo dobré, kdyby byla možnost provádět výpočty nad těmito hodnotami bez nutnosti zasahovat do zdrojového kódu aplikace. Bylo by nutné definovat nějaký skriptovací jazyk, ve kterém by uživatel daný výpočet popsal a podle kterého by následně aplikace vypočítala výsledné hodnoty metriky. Stálo by za zavážení, zda to činnost nevyčlenit do zvláštní služby, která by pak se službou pro správu

[19] http://github.com/xdrozd07/

dohod spolupracovala a poskytovala mu pouze výsledné hodnoty. Inspiraci by bylo možné hledat v [8].

Druhým možným rozšířením, které vyvstalo v průběhu vývoje je samotný princip sběru dat. Nyní si služba jednou za čas sáhne pro data a ta dále prezentuje uživateli ve formě reportu. Bylo by dobré, kdyby služba mohla tato data sbírat a kontrolovat je v reálném čase (nebo s malým zpožděním). Opět se jedná o námět na samostatnou webovou službu, se kterou by zde prezentovaná mohla kooperovat. Bylo by ovšem nutné provést analýzu, zda vůbec existují služby, u kterých by poskytovatelé byli ochotní zveřejňovat v reálném čase údaje o jejich běhu.

Posledním možným směrem dalšího vývoje je úprava klientské aplikace. Současné technologie nabízí stovky možností, jak učinit grafické uživatelské rozhraní přívětivějším pro uživatele, jak lépe prezentovat data a jak lépe umožnit práci se systémem. Tato část by vyžadovala neustálé zdokonalování a především reálné nasazení aplikace, neboť použitelnou úpravu GUI neleze provádět bez patřičné zpětné vazby od uživatelů.

Literatura

[1] Gartner Says Worldwide IT Spending on Pace to Reach $3.7 Trillion in 2013. *Gartner, Inc.* [online]. 2013 [cit. 2013-12-27]. Dostupné z: http://www.gartner.com/newsroom/id/2537815.

[2] Kolektiv autorů. *ITIL - výkladový slovník a zkratky v češtině*. Praha: itSMF Czech Republic, o.s., 2012. Dostupné z: http://www.itsmf.cz/uws/include/download.asp?file=/uws_files/publikace_ke_stazeni/itil_2011_czech_glossary_v2.0.pdf

[3] *ITIL 2011*. 1. vyd. Brno: Computer Press, 2012, 216 s. ISBN 978-80-251-3732-1.

[4] Cartlidge, A.; Hanna, A.; Rudd, C.: Úvodní přehled ITIL R V3. itSMF Czech Republic, o.s., 2007, ISBN 0-9551245-8-1

[5] CANNON, David. HP. *ITIL service strategy*. 2nd ed. London: TSO, 2011, xii, 483 s. Best Management Practice. ISBN 978-0-11-331304-4.

[6] HUNNEBECK, Lou. *ITIL service design*. 2nd ed. London: TSO, 2011, xi, 442 s. Best Management Practice. ISBN 978-0-11-331305-1.

[7] PALETA, Petr. *Co programátory ve škole neučí aneb Softwarové inženýrství v reálné praxi.* Vyd. 1. Brno: Computer Press, 2003. ISBN 80-251-0073-1.

[8] ASIT, Dan, Ludwig HEIKO a Pacifici GIOVANNI PACIFICI. Web service differentiation with service level agreements. *IBM developerWorks: IBM's resource for developers and IT professionals* [online]. 2003 [cit. 2014-01-02]. Dostupné z: http://www.ibm.com/developerworks/library/ws-slafram/

[9] LARMAN, Craig. *Applying UML and patterns: introduction to object-oriented analysis and design and iteractive development*. 3rd ed. New Jersey: Prentice-Hall, 2005, 703 s. ISBN 01-314-8906-2.

[10] GROSSMAN, Jeremiah. *XSS attacks: cross site scripting exploits and defense*. Burlington: Syngress, 2007, xiv, 448 s. ISBN 15-974-9154-3.

[11] HALFOND, W. G.; VIEGAS, Jeremy; ORSO, Alessandro. *A classification of SQL-injection attacks and countermeasures.* In: Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA. 2006. p. 13-15.

[12] CASTAGNETTO, Jesus. *Programujeme PHP profesionálně.* 2. oprav. a aktualiz. vyd. Překlad Ludvík Roubíček. Brno: Computer Press, 2004, xxiv, 656 s. ISBN 80-722-6310-2.

[13] JENDROCK, Eric. *The Java EE 5 tutorial*. 3rd ed. Upper Saddle River, NJ: Addison-Wesley, c2006, xli, 1304 p. ISBN 03-214-9029-0.

[14] PEAK, Patrick a Nick HEUDECKER. *Hibernate quickly.* Greenwich, CT: Manning, c2006, xxv, 425 p. ISBN 978-193-2394-412.

[15] *MySQL profesionálně: optimalizace pro vysoký výkon*. Vyd. 1. Brno: Zoner Press, 2009, 712 s. Encyklopedie webdesignera. ISBN 978-80-7413-035-9.