České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jan Greš**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Webový editor požadavků pro nástroj Enterprise Architect**

Pokyny pro vypracování:

Vytvořte webovou aplikaci, která bude umožňovat prohlížení, tvorbu a úpravu požadavků (a souvisejících elementů) evidovaných v databázi nástroje Enterprise Architect.
Zadání může být vedoucím práce rozšířeno o níže uvedené funkcionality v závislosti na míře složitosti vývoje základu aplikace.
Možná rozšíření:

A. Grafické zvýrazňování změn požadavků od určitého termínu či verze (inspirace z Microsoft Word).
B. Zobrazování a editace požadavků v přímo v diagramu.
C. Generování dokumentace (RTF/LATEX/,...)

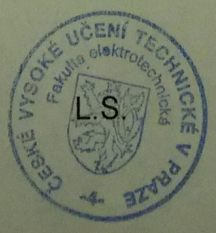
Zadání řešte projektovým způsobem[2], s využitím iterativního vývoje, pod BSD licencí a využitím jazyka UML a nástroje Enterprise Architect[1]. Veškerou dokumentaci projektu včetně počtu odpracovaných na jednotlivých úkolech/činnostech průběžně nahrávejte na FREE PUBLIC ASSEMBLA PROJEKT[4]. Při návrhu a implementaci uživatelského rozhraní klaďte důraz na jednoduchost a intuitivnost ovládání.

Seznam odborné literatury:

[1] Enterprise Architect, http://sparxsystems.com/
[2] Robert K. Wysocki: Effective Project Management: Traditional, Agile, Extreme
[3] http://www.lieberlieber.com/en/model-engineering/enarweb/product-overview/.
[4] https://www.assembla.com/catalog?type=public

Vedoucí: Ing. Martin Komárek

Platnost zadání: do konce letního semestru 2015/2016



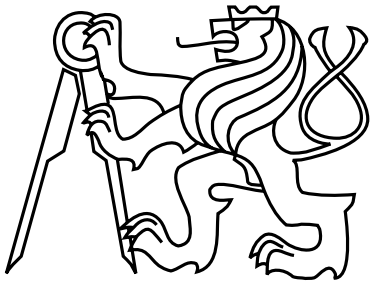
L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 12. 11. 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Webový editor požadavků pro nástroj Enterprise Architect

Jan Greš

Vedoucí práce: Ing. Martin Komárek

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

5. ledna 2015

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu práce Ing. Martinu Komárkovi za podnětné připomínky během konzultací a pevné vedení práce, díky čemuž jí bylo možné včas a na dobré úrovni dokončit.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Souhlasím, že práce bude uvolněna pod BSD licencí.

V Praze dne 5. 1. 2015 .....

Abstract

The main aim of this bachelor thesis is creating of web based application, providing requirement management for Enterprise Architect database. The application is starting as brand new independent project designated to another development. Emphasis is placed especially on simplicity and intuitive control.

The content of this document is complete project documentation generated gradually during iterative development. The work arises under the BSD license using UML and Enterprise Architect tool.

Abstrakt

Hlavním cílem této bakalářské práce je vytvořit webovou aplikaci, umožňující správu požadavků evidovaných v databázi nástroje Enterprise Architect. Aplikace vzniká jako zcela nový a samostatný projekt určený k dalšímu rozvíjení. Důraz je kladen především na na jednoduchost a intuitivnost ovládání.

Obsahem tohoto dokumentu je kompletní projektová dokumentace vznikající postupně během iterativního vývoje. Práce vzniká pod BSD licencí s využitím jazyka UML a nástroje Enterprise Architect.

Obsah

Kapitola 1

Úvod

1.1 Seznámení s projektem

Předmětem této bakalářské práce je vytvoření webového editoru pro požadavky k modelovacímu nástroji Enterprise Architect[1] (dále jen EA) od společnosti Sparx Systems.

Zmiňovaný nástroj slouží vývojovému týmu k postupnému modelování nově vyvíjeného počítačového softwaru prostřednictvím standardizovaných diagramů UML [8]. Tyto pojmy zde nebudou dále rozebírány, protože to není obsahem práce. Důležité je, že nástroj pracuje s mnoha prvky diagramu z nichž jedním typem jsou právě *požadavky*[2], na které se budeme dále zaměřovat. Celý projekt je pak uložen v lokálním databázovém souboru, který může být v korporátním prostředí nahrazen repozitářem ve skutečné sdílené databázi prostřednictvím intranetu nebo extranetu.

Samotný webový editor pak bude sloužit jako jakýsi externí nástroj, který s touto databází bude umět pracovat a modifikovat jí takovým způsobem, aby zůstala zachována její konzistence a integrita a veškeré změny byly okamžitě promítnutelné do projektu v nezávisle spuštěném nástroji EA.

1.2 Cíle a motivace

Hlavní motivací projektu je jednoduché, přehledné a intuitivní prostředí bez zbytečných pokročilých funkcí, odstiňující správu požadavků od zbytku projektu při zachování všech náležitých souvislostí. Další výhodou tohoto řešení je pak možnost správy jednotlivých požadavků z libovolného počítače prostřednictvím webového prohlížeče bez nutnosti instalovaného prostředí EA. Kupříkladu tedy někde v terénu, na cizím počítači, přes mobilní telefon apod.

[1] <http://www.sparxsystems.com/products/ea/>

[2] Jednotná zdokumentovaná potřeba (fyzická nebo funkční), kterou musí umět určitý produkt nebo proces naplnit. Více [25].

1.3 Cílová skupina

Cílovou skupinou pro toto řešení jsou pak především takoví členové vývojového týmu, kteří nemají příslušná oprávnění a pravomoce k manipulacím s jinými entitami projektu, než jsou samotné požadavky, nebo ti, kteří třeba jenom nemají dostatečné technické znalosti k práci se samotným vývojovým prostředím EA. Jedná se tedy především o lidi na vyšších řídících pozicích (kteří mají svůj zájem omezen pouze na manipulaci s požadavky) a nebo o lidi pro správu požadavků speciálně určené.

1.4 Metodika vývoje

Vývoj projektu bude probíhat iterativním způsobem. Práce bude podle Počátečního plánu projektu (viz kapitola 2) rozdělena na jednotlivé dílčí úseky nazvané iterace. Obsahem každé iterace bude samostatný cyklus analýzy, návrhu, především implementace a v neposlední řadě i testování a nasazení. Výsledkem každé iterace by měla být funkční aplikace s určitou omezenou funkcionalitou, což by mělo celý projekt posouvat neustále kupředu. Projekt tak bude vznikat po částech a postupně.

Kapitola 2

Počáteční plán projektu

2.1 WBS

Celý projekt byl pomocí technologie WBS [28] rozložen do následující struktury: Obrázek 2.1 a obrázek 2.2.

2.2 Iterace

Práce na projektu byla na základě rozkladu WBS rozdělena do následujících deseti iterací, které počtem zhruba pokrývají časové období určené k vypracování samotného projektu. Jednotlivé iterace by si se měly vzájemně zhruba rovnat podle náročnosti jejich dosažení. Předpokládaná doba realizace jedné iterace je zhruba 10 - 40 hodin práce v závislosti na vzniku možných komplikací.

- 1. iterace
  - seznámení se s projektem
  - analýza proveditelnosti
    - instalace prostředí EA
    - konfigurace prostředí EA pro databázový repozitář
    - analýza databáze a tabulek
- 2. iterace
  - analýza strukturálních vazeb v databázi
  - příprava vývojového prostředí
    - upgrade vývojového prostředí
    - nasazení frameworku
    - vytvoření prázdného projektu
  - prototyp projektu
    - konfigurace databáze

        * modelu požadavku (příp. dalších souvisejících entit)
        * vytvoření základní CRUD [13] manipulace s požadavkem

- 3. iterace
  - tvorba dokumentace
    * úvod do projektu
    * plánu projektu
  - autentizace
    * uživatelský model (příp. dalších souvisejících entit)
    * zabezpečení aplikace
    * autorizace
  - strukturování
    * model balíčků (příp. dalších souvisejících entit)
    * změna layoutu aplikace
    * implementace stromové struktury

- 4. iterace
  - grafické prvky
    * extrakce grafických prvků z EA
    * postprodukce grafických prvků
    * implementace grafických prvků do projektu
  - základní vzhled
    * změna layoutu aplikace
    * základní stylování
  - související elementy
    * analýza souvisejících elementů
    * vytvoření modelů pro jednotlivé elementy

- 5. iterace
  - implementace elementů
    * napojení elementů do aplikace
    * vhodná grafická úprava
    * prohlížeč elementů
    * možnost základní manipulace s elementy

- 6. iterace
  - manipulace s požadavky
    * přesouvání požadavků
    * verzování požadavků

        * accounting požadavků
    - další přídavné funkce a přidaná hodnota
        * drobná vylepšení

- 7. iterace
    - diagramy a grafy
        * analýza EA skriptů na generování grafů
        * příprava konzistentních dat pro grafy
        * zobrazení grafů
        * interakce s grafy

- 8. iterace
    - export
        * zvolení vhodného výstupu
        * příprava konzistentních dat pro export
        * implementace exportu
        * správa exportovaných souborů

- 9. iterace
    - další funkce
        * drobná vylepšení
    - překlad
        * překlad do češtiny
        * implementace přepínání lokalizace
    - tvorba dokumentace
        * testování funkčních požadavků
        * testování případů užití
        * testování vstupů
        * testování zabezpečení

- 10. iterace
    - opravy chyb nalezených při testování
    - rezerva

2.3 Rizika projektu

Po zhodnocení WBS a Prvotního plánu projektu jsem sestavil seznam následujících bodů, obsahující faktory, které jsou pro průběh projektu nějakým způsobem rizikové:

- **Analýza**

  - *Proveditelnost*

    Určité riziko se skrývá i v celé proveditelnosti projektu samotného. Je zapotřebí zanalyzovat databázi prostředí EA a zjistit, zda je vůbec možné ji externě modifikovat bez ztráty její integrity a zda je EA takovéto ochoten vůbec akceptovat. Nezdar by mohl ohrozit celý projekt.

- **Návrh**

  - *Špatný model databáze*

    Další riziko může vyvstat už při samotném návrhu, kdy se budou postupně rodit modely jednotlivých entit a objektů. Riziko zde vidím hlavně proto, že pracuji s již navrženým modelem databáze jednotlivé tabulky včetně jejich vztahů již existují. Modelování relačně mapovaných objektů (dále jen ORM [23]) je pak do jisté míry omezeno a především iterativní vývoj může způsobit, že se jednotlivé modely budou muset v průběhu celého projektu častěji předělávat, což by mohlo způsobit časové prodlevy.

- **Implementace**

  - *Nejasnost zadání*

    Mezi hlavní rizikové faktory projektu můžeme zařadit i implementaci tzv. „souvisejících elementů“. Toto slovní spojení pochází přímo ze zadání práce samotné a jeho interpretace v kontextu projektu je zcela nejednoznačná. Riziko tak spočívá ve správné identifikaci patřičných databázových struktur mimo hlavní entity požadavků a především jejich grafickém a pojetí, které musí být přehledně zvládnuté.

  - *Diagramy a grafy*

    Další kritický bod by mohlo být sestavení a zobrazení vizuálních diagramů a grafů, kde bude nutné nejprve zjistit, jak toto řeší EA a následně se pokusit jeho řešení převzít. Potencionálně získané algoritmy však budou muset muset býti výrazně ohnuty na míru této aplikaci. V případě neúspěchu se nabízí alternativa v podobě omezeného zobrazení pomocí technologií třetích stran, např. Google Charts[1].

Na všechny výše uvedená rizika bude během vývoje brán zvýšený zřetel, aby jejich dopad na výsledek byl pokud možno minimální.

2.4 Funkční požadavky

Systém bude především jednoúčelový a vyhotoven přesně na míru jeho zadání, které je sice samo o sobě již dost výmluvné, nicméně zde uvedu seznam předpokládaných funkčních požadavků na systém a i několik scénářů případu užití. Číslo v závorce udává předpokládanou prioritu jednotlivých požadavků.

[1] <https://developers.google.com/chart/>

1. **Přihlášení** (4)

   Systém bude umožňovat přihlášení. Nebude dostupný komukoliv a pro jeho používání bude nutné zadat jméno a heslo. Při jeho prvním spuštění se vytvoří sada přednastavených uživatelských účtů se standardními hesly.

2. **Uživatelské role** (2)

   Systém bude rozlišovat více druhů uživatelských rolí, kde každá role bude kaskádově obsahovat méně či více funkcí, než ta předchozí resp. následující. Více o rolích v samostatné sekci.

3. **Zobrazení diagramové struktury elementů** (5)

   Systém bude zobrazovat diagramovou stromovou strukturu elementů známou z prostředí EA, která bude interaktivní a umožňovat rozkliknutí libovolného objektu a vylistování si jeho přidružených požadavků. Tato struktura by měla obsahovat i základní grafické prvky a ikony.

4. **Prohlížení požadavků** (5)

   Systém bude umožňovat jednotlivé elementy požadavků otvírat a zobrazovat jejich veškeré podrobnosti.

5. **Úprava požadavků** (5)

   Systém bude umožňovat plnohodnotnou manipulaci s elementy požadavků a po otevření je tedy bude možné upravit. K tomuto účelu bude dostupný přehledný formulář poskytující různé přednastavené seznamy možných hodnot, načtené dynamicky přímo z databáze EA.

6. **Tvorba požadavků** (5)

   Systém bude umožňovat nový element požadavku také přidat a umístit do konkrétního balíčku. K tomuto účelu bude dostupný přehledný formulář poskytující různé přednastavené seznamy možných hodnot, načtené dynamicky přímo z databáze EA.

7. **Mazání požadavků** (5)

   Systém bude umožňovat již existující elementy požadavků i úplně odstranit z projektu. Na tuto operaci se systém nejprve zeptá pro potvrzení.

8. **Přesouvání požadavků** (3)

   Systém bude umožňovat i některé další pokročilé manipulace s požadavky jako například jejich přesun mezi jednotlivými objekty a balíčky, kopírování atp. Konkrétní akce budou upřesněny během tvorby projektu prostřednictvím sběru požadavků od zadavatele.

9. **Volitelné verzování požadavků** (1)

   Systém bude volitelně uchovávat historii všech provedených změn v požadavcích a bude schopen je graficky znázornit pomocí časových razítek na časové ose. Díky tomu bude možné se kdykoliv vrátit do libovolné předchozí verze a zároveň tak mít přehlednou informaci o jejich postupném vývoji.

Konkrétní podoba tohoto požadavku bude upřesněny během tvorby projektu prostřednictvím sběru požadavků od zadavatele.

10. **Volitelné grafické zobrazení elementů v diagramech** (1)

    Systém bude volitelně vizualizovat určité grafy a diagramy z prostředí EA, které budou zrovna spadat do kontextu aktuálně prohlížené entity. Tato funkcionalita by měla spoléhat na lokální scripty EA, které by měly poskytnout identický vzhled i mimo tuto aplikaci.

    Konkrétní podoba tohoto požadavku bude upřesněny během tvorby projektu prostřednictvím sběru požadavků od zadavatele.

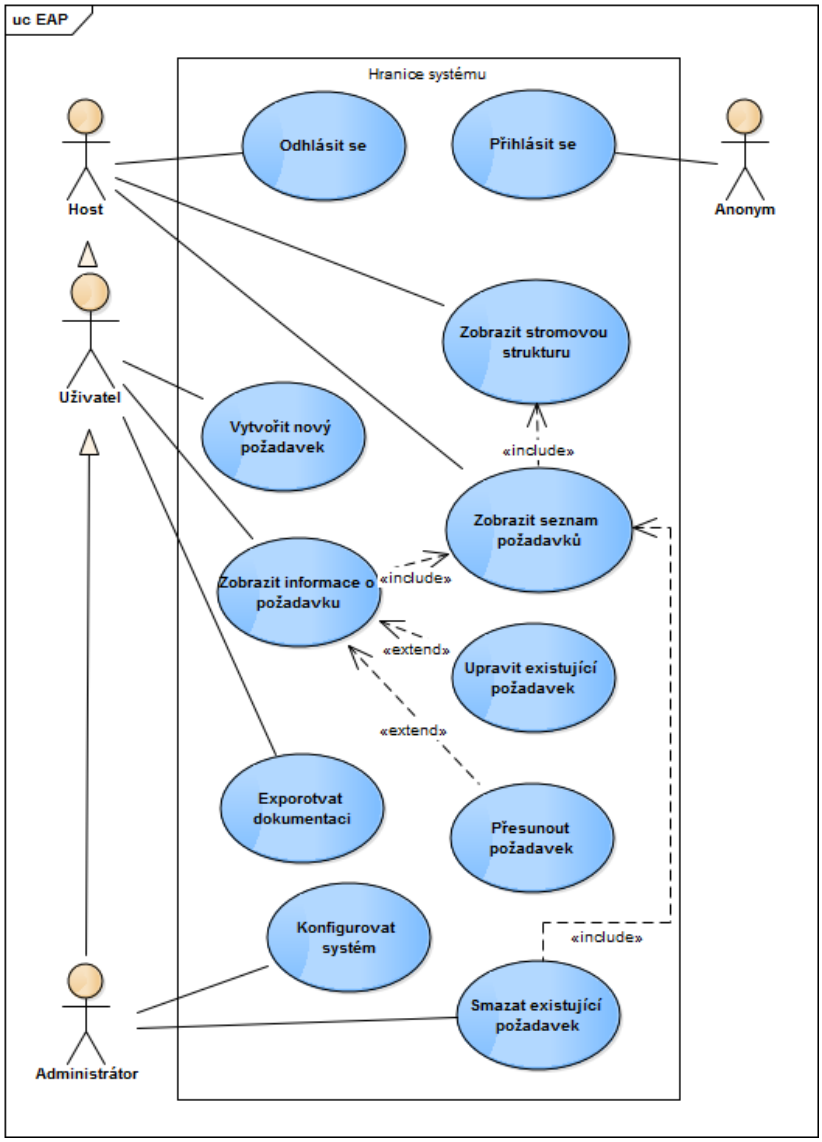
11. **Volitelný export požadavků pro tvorbu dokumentace** (1)

    Systém bude volitelně poskytovat funkcionalitu exportování všech dílčích požadavků do *plaintextového* LaTeXsouboru, který bude možné postprocesovat v tomto prostředí jako plnohodnotnou dokumentaci požadavků k danému projektu.

    Konkrétní podoba tohoto požadavku bude upřesněny během tvorby projektu prostřednictvím sběru požadavků od zadavatele.

2.5 Případy užití

Následující obrázek znázorňuje základní případy užití navrhovaného systému:



Obrázek 2.3: Případy užití

2.6 Scénáře případů užití

Následující tabulka demonstruje základní scénáře užití:

Jméno případu užití	Přihlášení do systému
Zúčastnění aktéři	Uživatel, Systém
Tok událostí	1. Uživatel zadá URL adresu do prohlížeče. 2. Systém zobrazí přihlašovací formulář. 3. Uživatel zadá své přihlašovací údaje (jméno a heslo) a stiskne tlačítko *Přihlásit se*. 4. Systém se pokusí se najít zadaného uživatele v databázi. 4.1. {Uživatel byl nalezen} Systém zahashuje vložené heslo. 4.2. Systém porovná zahashované heslo s hashí získanou z databáze. 4.2.1. {Hashe se rovnají} Systém založí uživatelské sezení s příslušnou rolí. 4.2.2. Systém zobrazí úvodní obrazovku se stromovým výpisem diagramové struktury. 4.2.3. Uživatel je přihlášen. Konec. 4.3. Uživatel není přihlášen z důvodu neplatnosti hesla. Konec. 5. Uživatel není přihlášen, protože jeho účet nebyl nalezen. Konec.
Vstupní podmínky	Uživatel má spuštěný webový prohlížeč, ale není na stránce systému. Rovněž není a nebyl přihlášen.
Výstupní podmínky	Uživatel je na úvodní stránce systému a je přihlášen.
Požadavky kvality	Systém nesmí přihlásit nikoho s neexistujícím jménem nebo neplatným heslem.

Jméno případu užití	Úprava existujícího požadavku
Zúčastnění aktéři	Uživatel, Systém
Tok událostí	1. Uživatel si vybere požadavek k editaci. 2. Uživatel klikne na tlačítko *Upravit* u tohoto řádku. 3. Systém dostane ID požadavku a načte jej z databáze. 4. Systém zobrazí formulář pro tvorbu a editaci požadavku. 5. Systém do zobrazeného formuláře vyplní načtené hodnoty. 6. Uživatel si vybere atributy, které by rád změnil. 7. Uživatel provede požadované změny. 8. Uživatel klikne na tlačítko *Uložit*. 9. Systém zaktualizuje všechny nové informace v databázi.
Vstupní podmínky	Uživatel je přihlášen a má otevřený příslušný balíček.
Výstupní podmínky	Požadavek je trvale změněn v databázi.
Požadavky kvality	Operace je systémem dokončena do 5 sekund.

2.7 Obecné požadavky

Následuje seznam kvalitativních požadavků:

1. **Systém bude kompatibilní s prostředím Enterprise Architect.**

   Systém jako takový bude v podstatě plugin pro aplikaci Enterprise Architect a proto jako hlavní obecný požadavek musí být vedena plná interkompatibilita tímto prostředím. Konkrétní nasazení bude testováno na verzi 11 Corporate Edition.

2. **Systém bude jednoduchý a spolehlivý.**

   Systém nebude obsahovat žádné zbytečné konstrukční složitosti a přídavné proprietární funkce. Bude upřednostňovat hlavně kvalitu před kvantitou, což znamená, důraz bude kladen především na primární funkcionalitu a její spolehlivost.

3. **Systém bude dále rozšiřitelný.**

   Celý systém bude možné dále libovolným způsobem rozšiřovat, čemuž přispěje hlavně přehledný a dobře strukturovaný zdrojový kód.

4. **Systém bude uživatelsky přívětivý.**

   Systém nebude obsahovat žádné zbytečné grafické prvky, které by zastávaly pouze dekorativní funkci. Měl by sice být graficky líbivý, avšak jednoduchý, čistý a přehledný,

aby jeho uživatel zbytečně netápal a vždy věděl jak dosáhnout svého cíle. Jinými slovy bude kladen extra důraz na jeho intuitivnost.

5. **Systém bude multiplatformní na straně serveru.**

   Systém bude možné provozovat na všech běžně provozovaných webových serverech pod různými operačními systémy jako např. Linux, Windows či Unix.

6. **Systém bude multiplatformní na straně klienta.**

   Systém bude možné používat na všech běžně používaných webových prohlížečích jako např. Internet Explorer[2], Mozilla Firefox[3], Opera[4] a další a to nezávisle na platformě.

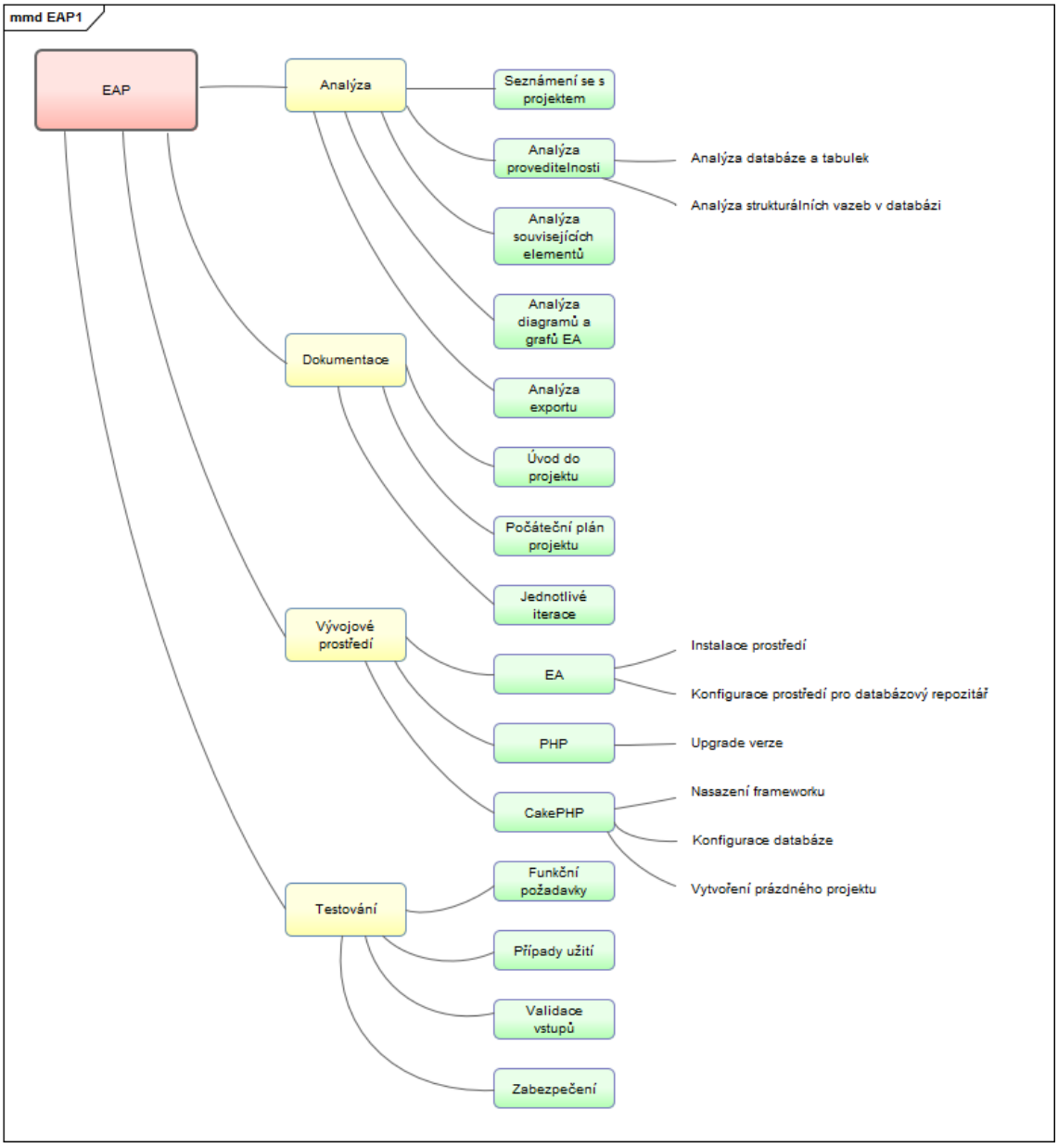
7. **Systém bude podporovat lokalizaci.**

   Systém bude napsán ve frameworku, který má silnou podporu lokalizace. Veškeré textové řetězce tedy budou sloužit jako lokalizační klíče pro dynamický překlad do libovolného jazyka. Systém bude vyhotoven v jazyce anglickém a přeložen do jazyka českého. Mezi těmito jazyky bude možné za chodu přepínat.

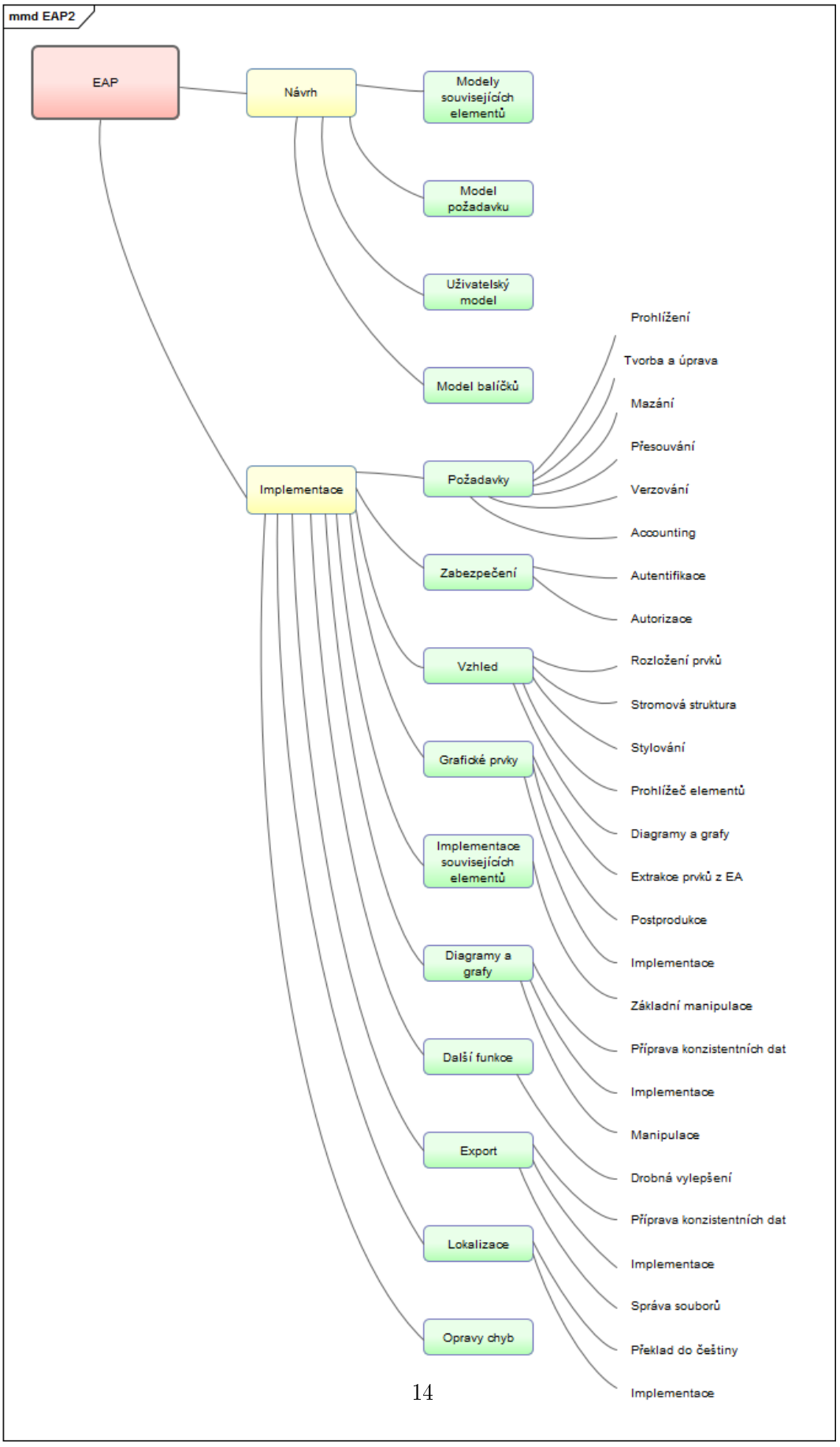
8. **Systém bude zabezpečený.**

   Systém bude díky použitému frameworku mimo jiné také bezpečný. To znamená, že by neměl umožnit přihlášení neautorizovaného uživatele.

[2] <http://windows.microsoft.com/cs-cz/internet-explorer/download-ie>
[3] <https://www.mozilla.org/cs/firefox/new/>
[4] <http://www.opera.com/cs/computer>



Obrázek 2.1: WBS - 1. část



Obrázek 2.2: WBS - 2. část

Kapitola 3

1. iterace

3.1 Zadání iterace

První iterace má za úkol především posbírat základní informace o projektu, podrobně se s nimi seznámit a hlavně vytvořit *Analýzu proveditelnosti* 3.2.2.

3.2 Analýza

3.2.1 Analýza případů užití

Prostřednictvím analýzy případů užití byly v systému identifikovány následující uživatelské role:

1. **Host (Guest)**

   Role bez jakýchkoliv dalších oprávnění. Jejím účelem je pouze možnost se do systému přihlásit a zase odhlásit. Dala by se charakterizovat jako read-only. Tedy jenom pro čtení. Takový uživatel může do všeho nahlížet, listovat tím, ale nemůže nic měnit.

2. **Uživatel (User)**

   Běžná uživatelská role umožňující takřka vše v systému dostupné. Jsou zde možné editace, tvorba nových požadavků, všechny pokročilé funkce, vizualizace, exporty atp.

3. **Administrátor (Admin)**

   Speciální správcovská role umožňující oproti běžnému uživateli především mazání elementů, správu uživatelů a pokročilou konfiguraci systému samotného.

3.2.2 Analýza proveditelnosti

Projekt Enterprise Architect Plugin (dále jen EAP), si klade za cíl vytvořit funkční plugin pro paralelní úpravu a správu požadavků v právě otevřeném aktuálním projektu prostředí EA prostřednictvím sdílené databáze obsahující repozitář s projektovými daty.

Předpoklad je, že EA využívá standardní databázový model vzájemně propojených tabulek, které budou volně přístupné (dotazovatelné) i mimo prostředí EA a struktura vnitřních dat bude v natolik čitelné a srozumitelné podobě, že jí bude velice snadné simulovat.

K tomuto ověření je však nejprve potřeba mít prostředí EA pro tento druh projektů správně nakonfigurované, protože databázové repozitáře nejsou jeho výchozím nastavením. (Ve výchozí instalaci se projekty ukládají do lokálních souborů, které představují malou souborovou databázi blíže nezkoumaného typu. Podrobný návod, jak tohoto nastavení správně docílit naleznete v sekci 3.4.1.

3.2.3 Analýza databáze a tabulek

Po úspěšném zprovoznění databázového repozitáře v prostředí EA je na čase zanalyzovat jeho databázovou strukturu. Databáze obsahuje několik málo tabulek s prefixem *usys* a zbytek drtivé většiny s prefixem *t_*. Celou její implementaci zajišťuje volně dostupné MySQL[1].

Výsledkem analýzy je dobrá zpráva a sice, že projekt EAP je možné provést přesně podle jeho úvodních předpokladů. Data v tabulkách jsou velice jasně strukturovaná a především v *plaintextové* formě, takže odpadá situace s případným dekódováním do čitelné podoby. Celá databáze je velmi silně textově orientovaná, což mi připadá dosti nezvyklé. Datové typy *varchar* a *text* zde vedle *int* skutečně převládají a drží z nepochopitelného důvodu i hodnoty jako jsou například časová razítka a různé přidružené vlastnosti objektů, které by mohly pomocí cizího klíče býti referencovány do separátních tabulek, které jsou zde dokonce přítomny!

Podle mého názoru tak tento stav vypovídá o tom, že celá funkcionalita databázových repozitářů byla do prostředí EA zabudována až mnohem později, kdy již byla datová struktura pevně dána. Přes to, že indexů je v databázi skutečně dost a některé klíče jsou dokonce přes vícero sloupců, nenašel jsem žádnou referenční integritu, takže výhody enginu InnoDB[2] vůbec nevynikají.

3.2.4 Definice požadavku

Po důkladném prozkoumání databáze a identifikace *požadavku* samotného (tak, jak jej chápe EA) jej nyní můžeme přesně definovat.

Prostředí EA pracuje se širokou škálou různých entit ať už se jedná o elementy standardu UML či jiných modelovacích jazyků. Entitou pak může být vlastně cokoliv, např. třeba *aktér*, *diagram*, *třída*, *případ užití* a nebo konečně i samotný *požadavek*. Všechny tyto entity má systém v databázi uložené jako jednotlivé objekty různých typů. A *požadavek* je jedním z nich.

Požadavek je ale v celém ekosystému prostředí EA nejenom typ objektu, ale současně i jednice z množiny atributů, k jednotlivým objektům variabilně přidružená. Jednodušeji

[1] <http://www.mysql.com/>
[2] Typ ukládání dat v MySQL.

řečeno pak většina objektů (technicky vzato každý) může obsahovat množinu těchto dalších *požadavků* (budeme jim říkat *požadavky na objekt*), což jsou v podstatě takové textové poznámky k dané entitě (či objektu, chcete-li) a její problematice. Atributy mají tyto *požadavky na objekt* pak velice podobné těm, jako u samotného *požadavku*, jen mírně redukované.

3.2.5 Výsledek analýzy

Pro projekt EAP je výsledný stav velice uspokojivý, protože není nutné z databáze načítat a spojovat veliké množství tabulek k získání potřebného vzorku informací. Samotný objekt *požadavku na objekt* je uložen v tabulce *t_objectrequires* a obsahuje všechny potřebné atributy a vlastnosti. Každý takový požadavek však musí příslušet nějakému diagramovému objektu (mezi které patří i obyčejný *požadavek*), který reprezentuje tabulka *t_object* skrze cizí klíč *Object_ID*. Jednotlivé objekty jsou pak třízeny do stromové struktury prostřednictvím tzv. *balíčků*, které nalezneme v tabulce *t_package* skrze cizí klíč *Package_ID*. Tyto tři tabulky tak budou pro celý projekt EAP stěžejní a jim věnujeme největší pozornost.

I když jsou možná dva typy požadavků dosti zavádějící skutečností, nedá se s tím bohužel nic dělat. Tato struktura je dána návrhem databáze autorů EA a musíme jí tedy respektovat. Projekt EAP se bude snažit nějakým způsobem umožnit editaci jak obyčejných *požadavků*, tak i přidružených *požadavků na objekt*.

3.3 Návrh

3.3.1 Zvolené prostředí a technologie

Přesto, že prostředí EA podporuje nespočet typů samotné databáze, celé řešení bude vyvíjeno a testováno ve volně šiřitelném MySQL [9]. Tento typ databáze byl vybrán pro jeho početnou rozšířenost v daném segmentu nekomerčních a polokomerčních aplikací. Mimo jiné také pro jeho snadnou dostupnost a mohutnou uživatelskou základnu [10] poskytující brilantní podporu při řešení nejrůznějších problémů.

Aplikace jako taková bude uživateli dostupná jako webová stránka dostupnou prostřednictvím sítě *Internet* standardním protokolem HTTP [18].

Samotnou implementaci projektu pak bude zastřešovat framework CakePHP [1], který určil i vývojovou platformu a programovací jazyk PHP [4]. Výběr programovacího jazyka tak netradičně podlehl zvolenému frameworku, který jsem vybral díky jeho skvělému objektovému mapování na databáze, což se pro tento typ řešeného projektu velice dobře hodí. Dalším plusem jsou pak ještě četné zkušenosti a praxe v samotném frameworku, což mi umožní projekt implementovat daleko efektivněji, než kdybych pracoval s nějakými neznámými technologiemi, které je nutné nejprve studovat. Díky CakePHP bude také mimo jiné v budoucnu velice snadné celou databázovou vrstvu (při zachování stávající struktury objektů) zmigrovat na jiný typ databáze, kupříkladu PostgreSQL[3] či dokonce SQLite[4].

[3] <http://www.postgresql.org/>
[4] <http://www.sqlite.org/>

Na straně serveru bude aplikace určená a testovaná především pro běhové prostředí Apache [2] a to jak na platformě Microsoft Windows [7], tak na platformě Linux Ubuntu [6]. Systém bude produkovat standardní HTML5 [30] výstup stylovaný kaskádovými styly CSS3 [29] a doplněn JavaScriptem [31].

3.4 Nasazení

3.4.1 Konfigurace EA pro databázový repozitář

Výchozím bodem tohoto postupu je platforma Windows s instalovaným a plně funkčním prostředím EA verze 11 ve výchozím nastavení a s funkční databází MySQL verze 5.6.20. Návod bude velice podobný, ne-li přímo stejný, i pro jiné verze, avšak nebyl na nich otestován. Instalace těchto programů není předmětem tohoto návodu. Prostředí EA je plně komerční produkt, kterého jsem nabyl skrze univerzitní licenci, zatímco databázový server MySQL je volně dostupný ke stažení na stránce <http://www.mysql.com>.

- Tvorba repozitáře.

  Ať už se to zdá divné či ne, samotný repozitář je nutné vytvořit v databázi ručně. Společnost Sparx Systems naštěstí dodává plně funkční SQL soubory obsahující kompletní databázové schéma. Můžete si je stáhnout z jejich webu, popřípadě budou přiloženy k projektu ve složce Install jako buď:

  `MySQL_MyISAM_EASchema.sql`

  Nebo:

  `MySQL_InnoDB_EASchema.sql`

  Soubory jsou dva v závislosti na použitém databázovém enginu. Přesto, že mám kvůli rychlosti raději MyISAM, jsem však musel nakonec instalovat InnoDB verzi, protože ta předchozí měla nějaké nestandardní požadavky na velikost datových typů, které engine ve výchozím nastavení nebyl schopen akceptovat. To stejné doporučuji i Vám, získáte tak integritně daleko silnější sadu tabulek.

  Jen pro úplnost zde ještě uvedu, že každý projekt má svojí vlastní databázi. Nejedná se tedy o jednu obří databázi, co pojme několikero projektů, ale pro každý nový EA projekt musíte ručně vytvořit novou a prázdnou SQL databázi libovolného jména s kódováním *utf8_general_ci*. Pro naše potřeby je to databáze nazvaná pochopitelně *ea*. Jakmile tuto databázi vytvoříte, stačí v ní provést všechny příkazy obsažené v SQL souboru a to buď přes nástroj MySQL Workbench[5] jejich vykopírováním ze souboru a nebo velice snadno skrze importovací funkci webového rozhraní phpMyAdmin[6], které

[5] <http://www.mysql.com/products/workbench/>
[6] <http://www.phpmyadmin.net/>

pomohlo i mě. Pokud se všechny tabulky vytvořily úspěšně a nic nezahlásilo chybu, repozitář je úspěšně vytvořen.

Podrobnější návod v anglickém jazyce viz <http://www.sparxsystems.com/enterprise_architect_user_guide/9.2/projects_and_teams/createanewmysqlrepository.html>.

- Instalace a konfigurace ovladače ODBC.

  Prostředí EA využívá pro spojení s databází standardizovaný ovladač (nebo také konektor) ODBC [22]. Použil jsem aktuální ve verzi 5.3.4. Na verzi ale pochopitelně nezáleží, pokud je vyšší než 5.1.5, což uvádí přímo výrobce EA. Co je ovšem důležité si uvědomit je, že nezávisle na OS je prostředí EA 32-bitovou aplikací. Zkušeným uživatelům to přijde automatické, ale pro správné fungování je nutné stáhnout příslušnou 32-bitovou verzi ovladače ODBC. Aktuální ovladač naleznete na adrese <http://dev.mysql.com/downloads/connector/odbc/>. Doporučuji verzi v MSI [20] balíčku pro snadnější instalaci.

  Po instalaci už jen stačí tento ovladač přidat do zdrojů dat v ovládacím panelu ODBC. Opět musíme použít 32-bitový ovládací panel, který nalezneme na cestě:

  ```
  C:\Windows\SysWOW64\odbcad32.exe
  ```

  V záložce *Uživatelských DSN* klikneme na tlačítko *Přidat* a vyhledáme námi instalovaný ovladač ODBC v Unicode[7] verzi. Do zobrazivší tabulky pak už jen vyplníme nějaké jméno, popis a hlavně přihlašovací údaje k databázovému serveru. (Ve výchozím nastavení se připojujeme přes *TCP/IP k localhostu skrze port 3306*.) Po vyplnění jména a hesla se nám pak načte seznam dostupných databází a my můžeme vybrat tu naší s instalovaným repozitářem a vše otestovat tlačítkem *Test*. V rozšířeném nastavení (tlačítko *Details »*) pak ještě zaškrtneme *Allow big result sets* v kartě *Connection* a *Return matched rows instead of affected rows* v kartě *Cursors/Results*. Vše uložíme a je hotovo.

  Podrobnější návod v anglickém jazyce viz <http://www.sparxsystems.com/enterprise_architect_user_guide/9.2/projects_and_teams/setupmysqlodbcdriver.html>.

- Napojení se na repozitář z EA.

  Když už máme repozitář vytvořený a ovladač ODBC nastavený, můžeme už konečně spustit prostředí EA. V dialogovém okně, kde normálně vybíráme otevíraný projekt zvolíme nabídku *Connect to Server -> Connection Wizard*. Z otevřené nabídky pak vybereme zprostředkovatele *Microsoft OLE DB Provider for ODBC Drivers* a klikneme na tlačítko *Další »*. Zdroj dat pak vybereme pomocí jeho názvu z dostupného výběrového pole. Opět budeme muset zadat přihlašovací jméno a heslo k serveru a vybrat správný katalog neboli databázi. Vše můžeme otestovat tlačítkem *Testovat připojení* a v případě úspěchu potvrdit uložit. V nabídce posledních otevíraných projektů

[7] Typ kódování znaků a textu.

nám nyní přibyla nová položka s cestou k souboru (*Path*) jako *[MySQL repository]* a kdykoliv vybereme tuto možnost a projekt otevřeme, vše se nám načítá a ukládá do sdíleného databázového repozitáře.

Podrobnější návod v anglickém jazyce viz <http://www.sparxsystems.com/enterprise_architect_user_guide/9.2/projects_and_teams/connecttomysql.html>.

3.5 Zhodnocení

Výsledkem první iterace je tedy zjištění, že projekt EAP může bez jakýchkoliv komplikací pokračovat přesně tak, jak byl na začátku navržen. Paralelní úprava MySQL databáze je zcela bezproblémová, prostředí EA konzistentní změny reaguje pozitivně a především okamžitě. Konkrétně každá změna provedená nezávisle v databázi je do prostředí EA aplikována bezprostředně po interakci s daným prvkem.

Na jednom projektu tak může pracovat až několikero nezávislých lidí, kteří budou prostřednictvím aplikace EAP modifikovat a spravovat požadavky na jednotlivé objekty, což se ihned promítne do prostředí EA všem ostatním (a především výkonným) členům týmu.

Kapitola 4

2. iterace

4.1 Zadání iterace

Druhá iterace má za úkol sestavit podrobný plán projektu 2, připravit plně funkční vývojové prostředí, nastavit framework a vyhotovit fungující prototyp projektu se základními CRUD operacemi s objektem požadavek.

4.2 Analýza

4.2.1 Vývojové prostředí

Před započetím samotné implementace je zapotřebí mít plně funkční vývojové prostředí. Projekt EAP bude využívat frameworku CakePHP a proto je nutné mít instalovanou minimální verzi PHP verze 5.2.8. Starší verze nepodporují všechny potřebné funkcionality. Vývoj bude probíhat na serveru Apache verze 2.2.25 a PHP 5.3.29 pod platformou Windows. Jako textový editor zdrojového kódu jsem zvolil svůj oblíbený volně dostupný program Notepad++[1], který zvýrazňuje syntaxi mnoha známých programovacích jazyků, podporuje dobré formátování a nabízí dokonce i našeptávání parametrů standardních funkcí. Navíc je plně *plaintextový*.

4.2.2 Framework

Pro vývoj aplikace byl zvolen framework CakePHP a to především z důvodu, že má velice silnou podporu pro objektové mapování na databázi, které je velice snadné, rychlé a intuitivní. Tento framework lze získat na oficiálních stránkách výrobce <http://www.cakephp.org>. Pro projekt EAP je použita aktuální stabilní verze 2.5.5.

Pro jeho úspěšnou funkčnost není zapotřebí mnoho, kromě již zmíněné minimální verze PHP jsou už zapotřebí pouze práva k zápisu do složky *tmp* a správně nastavený engine cachování[2].

[1] <http://notepad-plus-plus.org/>
[2] Ukládání dat do vyrovnávací paměti.

Podrobnosti o konfiguraci frameworku naleznete v sekci 4.5.1.

4.3 Návrh

4.3.1 Architektura aplikace

Jak už přímo z podstaty objektového frameworku CakePHP vyplývá, celá aplikace EAP bude využívat silně objektový model a především tak architekturu MVC [21], kterou tento framework přesně kopíruje.

Nemá zde cenu pojem MVC vysvětlovat nijak do hloubky, ale pojďme se spíše podívat na to, jakým způsobem ho využívá CakePHP, aby byla celá technická dokumentace srozumitelnější a čitelnější.

Model

Modely jsou v CakePHP jakési stavební bloky. Každý model je namapován na určitou tabulku v databázi, kterou reprezentuje. Můžeme zde dále konfigurovat různá validační omezení, vztahy s jinými modely a pochopitelně i další pomocné metody pro práci s daným objektem v lokálním kontextu.

Controller

Controllery jsou v CakePHP místo, kde se obvykle objeví nejvíc zdrojového kódu, neboť je zde většina aplikační logiky. Controllery mapují modely (avšak ne každý model má svůj controller) a reprezentují v podstatě jména pro dostupné stránky. Každá stránka či podstránka v CakePHP má pak adresu v následujícím tvaru: *

```
app/Controller/action/param1/param2/param3/[...]
```

Jsou to v podstatě takové handlery pro webové požadavky. Podle názvu controlleru se načte požadovaná modelová struktura a spustí se požadovaná akce (action) již se předá zbytek parametrů. Programátor pak určí, co se za daných okolností má stát.

View

View jsou v CakePHP klasické CTP[3] šablony obsahující jako jediné víceméně čisté fragmenty HTML kódu s aktivními výstupy z PHP. Tyto šablony obsahují mnoho tříd pomocníků (tzv. Helpery) pro generování předpřipravených odkazů, obrázků, formulářů a jiných běžně využívaných struktur, takže je práce s nimi velice snadná a programátor je co nejvíce odstíněn od samotného HTML kódu.

Ve výchozím chování má každý controller složku svých view, kde je pro každou akci dostupný jeden takový. Toto chování se ale dá silně přetížit a modifikovat prakticky libovolným způsobem. View se dají mezi sebou navzájem volat a vkládat a obsahují i několik paralelních výstupů.

[3] Jedná se o příponu souboru.

4.3.2 Návrh modelu požadavku

K dosažení základních CRUD operací nad požadavkem nám bohatě postačí definovat si tento požadavek jako samostatný model. V našem případě se tedy jedná o tabulku *t_objectrequires* známou z předchozí iterace. Vše co musíme udělat je vytvořit pro model třídu a uložit do souboru *Model/Requirement.php*:

```
<?php
    App::uses('AppModel' , 'Model');

    class Requirement extends AppModel {
        public $displayField = 'Requirement';
        public $primaryKey = 'ReqID';
        public $useTable = 't_objectrequires';
    }
?>
```

Model *Requirement* zvolíme jako klíčový, protože se kolem něj točí jádro zadání celého projektu. Většina aplikační logiky tedy bude v jeho controlleru, který nastavíme v konfiguračním souboru routování *Config/router.php* následovně:

```
Router::connect('/' , Array('controller' => 'Requirement' , 'action' => 'index'));
```

V této fázi vývoje tak máme zatím tři modely:

Requirement

Základní model požadavku obsahující všechny potřebné atributy k jeho zobrazení a manipulaci.

RequirementType

Výčtový model pro různé typy požadavků. Tento modle přímo vyplynul z již definované databázové struktury projektového repozitáře EA.

RequirementStatus

Výčtový model pro různé stavy požadavků. Tento modle přímo vyplynul z již definované databázové struktury projektového repozitáře EA.

4.4 Implementace

4.4.1 Implementace controlleru požadavku

V prototypu aplikace bude controller požadavku obsahovat tři základní akce mapující čtveřici operací CRUD. Dovolil jsem si zde totiž spojit akce CREATE a UPDATE, aby byla aplikace kompaktnější. Největší výhoda ale spočívá se sdíleným view se samotným formulářem, který je až na výjimky, jak pro tvorbu nového požadavku, tak pro editaci, prakticky identický.

Máme tedy následující akce:

index()

Výchozí akce vypisující seznam všech dostupných požadavků.

edit(id)

Akce editující již existující požadavek při předání jeho ID parametrem. V případě, že je jako ID zvolena hodnota 0, formulář se načte prázdný a uložení způsobí tvorbu nového požadavku.

delete(id)

Akce mazající již existující požadavek, jehož ID musí být předáno parametrem.

4.5 Nasazení

4.5.1 Konfigurace CakePHP

Výchozí konfigurace CakePHP předpokládá umístění celého frameworku v kořenovém adresáři WWW serveru takto:

```
/cakephp/
```

Samotná aplikace je pak standardně dostupná v podložce *app* a cesta je tedy následující:

```
/cakephp/app/
```

Co když ale chceme na serveru provozovat více aplikací pod jedním frameworkem? Stačí upravit soubor *webroot/index.php* a definovat správnou cestu ke knihovnám frameworku:

```
Define('CAKE_CORE_INCLUDE_PATH' , ROOT.DS.'cakephp'.DS.'lib');
```

Aplikaci pak můžeme umístit do libovolné složky na serveru. V našem případě pak velice pohodlně jako:

```
/eap/
```

4.5.2 Konfigurace databáze v CakePHP

Framework CakePHP je silně objektový a na bez výjimky všechno využívá třídy. Není tedy divu, že i konfigurace databáze je jako samostatná třída. My se nyní připojíme na již dříve vytvořenou databázi *ea*. Následuje zdrojový kód konfigurace umístěné jako *Config/database.php*:

```
<?php
    class DATABASE_CONFIG {
        public $default = Array(
            'datasource' => 'Database/Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'login' => 'root',
            'password' => 'pass',
            'database' => 'ea',
            'prefix' => '',
            //'encoding' => 'utf8',
        );
    }
?>
```

4.6 Zhodnocení

Výsledkem druhé iterace je základní návrh a architektura vyvíjené aplikace, plně funkční a nakonfigurované vývojové prostředí včetně nového projektu, ale především pak funkční prototyp samotné aplikace. Ten již umí pracovat s databázovou strukturou prostředí EA a provádět na ní všechny čtyři základní operace CRUD. Výsledky jsou ihned promítnuty do testovacího projektu EA.

Kapitola 5

3. iterace

5.1 Zadání iterace

Třetí iterace má za úkol veškeré úsilí zaměřit především na implementaci. Aplikace by měla mít své zabezpečení a podporovat jak autentizaci, tak autorizaci na základě uživatelských účtů. Jako hlavní úkol je pak zadáno vytvoření modelu balíčku a vizualizace stromové struktury jednotlivých objektů. Aplikace by rovněž měla získat první náznaky vzhledu.

5.2 Analýza

5.2.1 Zabezpečení

Systém bude díky použitému frameworku mimo jiné také bezpečný. To znamená, že by neměla nastat situace, kdy by do systému pronikl nepovolaný útočník mimo použití hrubé síly či sociálního inženýrství. Bude kladen důraz na bezpečnost databázových dotazů a nebude tedy umožněno provést tzv. *SQL injekci*[1] či například *XSS*[2] a jiné druhy běžně známých útoků na webové stránky.

Při zevrubném průzkumu databázové reprezentace repozitáře projektu EA nebyla nalezena žádná tabulka uchovávající jakékoliv informace o uživatelích či jejich přístupech k projektovým datům. Koncepce bezpečnosti je tak postavena pouze na uživatelských přístupech do databáze samotné. Jinak řečeno s projektem může z prostředí EA pracovat kdokoliv, kdo má správně nakonfigurovaný ovladač ODBC.

Tento způsob zabezpečení však pro webovou aplikaci není vhodný a budeme jej muset pro projekt EAP vyřešit externě a sami.

[1] Technika provedení neautorizovaného SQL dotazu prostřednictvím běžně dostupného vstupu.
[2] Technika spuštění neautorizovaného JavaScriptového kódu.

5.2.2 Stromová struktura

Dalším zkoumáním databázové struktury bylo zjištěno, že každý požadavek se nutně váže k určitému objektu a ty jsou dále třízeny do jednotlivých balíčků o neomezené hloubce stromové struktury. Ta by měla být plně dodržena i v aplikaci EAP.

Jediným úskalím zde je fakt, že požadavky mohou být v prostředí EA přidruženy i k balíčku samotnému. Databázová reprezentace však na toto nepamatuje a každý takovýto balíček pak musí být objektově zdvojen a obsahovat instanci nejen objektu *t_package*, ale i *t_object*.

To by ještě nebylo to nejhorší, kdyby reference na předka nebyl nadřazený balíček samotného balíčku, ale intuitivně onen balíček samotný, jehož tento objekt reprezentuje. Dostáváme se tedy k paradoxní situaci, kde ve stromové struktuře na stejné úrovni hloubky máme pod stejným identifikátorem dvě entity. Balíček a jeho objekt, na který jsou navázány jeho požadavky.

Do samotné implementace to pak zanáší jednu zásadní inkonzistenci. Otevřeme-li si určitý balíček a necháme si vypsat všechny v něm na přímo obsažené požadavky, nebudou mezi nimi požadavky týkající se balíčku samotného. Ty nalezneme o úroveň výš.

5.3 Návrh

Uživatelský model

Protože výchozí podoba databázové reprezentace repozitáře projektu prostředí EA neřeší žádné uživatelské přístupy, je nutné toto zajistit svépomocí. Pro tyto účely jsem si dovolil zasáhnout přímo do databáze a vytvořit si zde vlastní tabulku *e_users*, která bude uchovávat id, jméno, MD5[3] hashi[4] hesla a roli každého uživatele. Kompletní schéma všech případných vlastních tabulek bude dostupné ve složce *Install*. Tabulka uživatelů bude tedy umístěna v souboru *Install/e_users.sql*. Pro představu zde uvádím obsah souboru:

```
CREATE TABLE e_users (
    id      int NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name        varchar(32) NOT NULL,
    password    varchar(32) NOT NULL,
    role        int NOT NULL DEFAULT 1
) ENGINE = InnoDB;

INSERT INTO e_users VALUES
    (1 , 'Admin' , MD5('admin') , 0),
    (2 , 'User' , MD5('user') , 1),
    (3 , 'Guest' , MD5('') , 2);
```

[3] Hashovací algoritmus.
[4] Minimalizovaný otisk vstupních dat.

Jak je z SQL dotazů patrné, jedná se o vytvoření databázové tabulky a v právě zvolené databázi a její naplnění výchozími hodnotami.

V samotné implementaci pak aplikace sama zajistí, aby se při detekci chybějících tabulek tyto tabulky samy doinstalovaly. Uživatel EAP pak nic z tohoto nemusí řešit, vše se děje plně automaticky na pozadí. V ideálním případě se pak vše potřebné nainstaluje pouze jednou, při prvním spuštění aplikace, načež na toto systém upozorní a zveřejní výchozí přihlašovací jména a hesla do systému.

5.3.1 Přehled modelů

Projekt tedy bude dále rozšířen o následující modely:

User

Model sloužící frameworku CakePHP jako autentifikační. Na základě něj bude prováděno přihlašování a udílení přístupů. Kromě sloupců id, jména, hesla a role zde již nejsou žádné další, protože klademe důraz na jednoduchost a cokoliv dalšího by bylo zbytečné.

Entity

Klíčový model představující libovolný objekt. Protože slovo *Object* je v mnoha programovacích jazycích vymezeno jako klíčové, bylo zvoleno jméno *Entity*. Tento model obsahuje další modely *Requirement* s kardinalitou 0..N.

Pro představu implementační třída napsaná v CakePHP:

```
<?php
    App::uses('AppModel' , 'Model');

    class Entity extends AppModel {
        public $displayField = 'Name';
        public $primaryKey = 'Object_ID';
        public $useTable = 't_object';

        public $hasMany = Array(
            'Requirement' => Array(
                'className' => 'Requirement',
                'foreignKey' => 'Object_ID'
            )
        );
    }
?>
```

Package

Přidružený model balíčku představující jednotlivé větve stromové struktury. Tento model obsahuje jak jednotlivé listy (objekty typu *Entity*) s kardinalitou 0..N, tak vazbu sám na sebe s kardinalitami 0..N a 0..1.

Opět uvádím zdrojový kód z implementace. Všimněte si zakomentované vazby 0..1, která není pro použití v tomto projektu žádoucí. Data budeme číst pouze směrem od kořene k listům a nikoliv naopak:

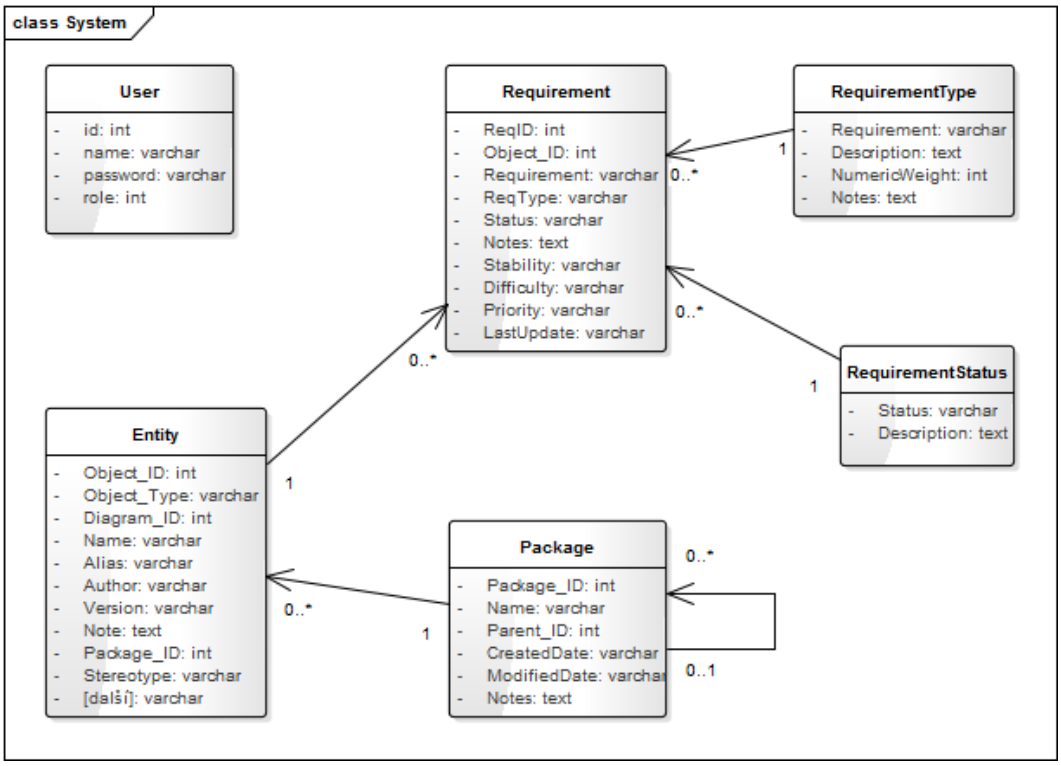
```
<?php
    App::uses('AppModel' , 'Model');

    class Package extends AppModel {
        public $displayField = 'Name';
        public $primaryKey = 'Package_ID';
        public $useTable = 't_package';

        public $hasMany = Array(
            'Children' => Array(
                'className' => 'Package',
                'foreignKey' => 'Parent_ID'
            ),
            'Entity' => Array(
                'className' => 'Entity',
                'foreignKey' => 'Package_ID',
                'conditions' => Array(
                    'Name IS NOT NULL',
                    'SUBSTRING(Name , 1 , 1) != \'$\''
                )
            )
        );

        /*public $belongsTo = Array(
            'Parent' => Array(
                'className' => 'Package',
                'foreignKey' => 'Package_ID',
                'associativeKey' => 'Parent_ID'
            )
        );*/
    }
?>
```

Takto pak vypadá dosavadní diagram tříd:



Obrázek 5.1: Diagram tříd 3. iterace

5.4 Implementace

Co se samotné implementace týče, zde nám došlo k drobnému vylepšení původního controlleru a především k přidání dalších dvou. Nyní se podíváme podrobněji na ně i jejich metody.

5.4.1 RequirementController

Stále klíčový controller celé aplikace. Nyní rozšířen o novou metodu *isAuthorised(user)*, která se stará o udílení přístupů k jednotlivým akcím na základě oprávnění přidruženým k jednotlivým uživatelským rolím.

Změny se pak dočkala především metoda *index(package , object)*, která dostala dva nové parametry a nově tedy umožňuje vylistování požadavků dle různých kritérií. Buď tuto metodu zavoláme bez parametrů a (jako v předchozí iteraci) vypíše úplný seznam požadavků celého projektu nebo je můžeme dále filtrovat. V případě volání s jedním parametrem (ID balíčku) se vypíší pouze takové požadavky, které se týkají objektů v tomto balíčku obsaženém. Pozor: Nevypíší se však požadavky týkající se balíčku samotného – viz poznatky z předchozí iterace. V případě volání se dvěma parametry (ID balíčku a ID objektu) se pak vypíší pouze ty požadavky, které jsou navázány přímo na konkrétní objekt obsažený v daném balíčku (tedy nejpřísnější filtr).

5.4.2 UsersController

Nový controller, který zatím slouží pouze k autentizace. Obsahuje metody *login()* a *logout()*. Obě jsou bez parametrů, protože o veškeré procesní záležitosti samotného přihlášení a odhlášení se nám krásně postará framework. Na tyto stránky je také uživatel automaticky přesměrován, pokud nemá dostatečná oprávnění.

5.4.3 AppController

Výchozí controller celé aplikace zde byl přítomný už od začátku, ale byl zatím prázdný. Jedná se o nadtřídu všech ostatních controllerů a platí tedy pro celou aplikaci. Nyní zastává v celém projektu důležitou úlohu. Jednak v něm je konfigurováno zabezpečení a to následujícím způsobem:

```
public $components = Array(
    'Session',
    'Auth' => Array(
        'authenticate' => Array(
            'EAP'
        ),
        'loginRedirect' => Array('controller' => 'requirement' , 'action' => 'index'),
        'logoutRedirect' => Array('controller' => 'users' , 'action' => 'login'),
        'authError' => 'This system require login!',
        'authorize' => Array('Controller')
    )
);
```

Všimněme si autentifikačního modulu EAP, který na základě zadaného jména vyhledá uživatele v databázi, porovná zahashované zadané heslo s jeho uloženou hashí a vrátí objekt přihlášeného uživatele v případě úspěchu. V případ neúspěchu vrací null[5].

Dále tu máme metodu *beforeFilter()*, která je volána před iniciací každého controlleru a ta se stará jednak o správnou integritu databáze (v případě prvního spuštění doinstaluje potřebné tabulky) a jednak o všudypřítomné načtení stromové struktury balíčků. Zde si dovolím poznámku aktuálního omezení aplikace. Protože jsou všechny SQL dotazy vytvářeny automaticky, musí být explicitně a pevně nastaven rekurzivní limit hloubky spojování tabulek. Pro tuto iteraci je zde limitní hloubka úrovně 5, ale toto téma je ponecháno k přehodnocení v dalších iteracích.

Následují pak dvě statické metody *findPackageById(packages , id)* a *findEntityById(packages , id)*, jejichž názvy jsou myslím dosti výmluvné. Ani zde pak nechybí nadřazená metoda *isAuthorised()*, která se stará o zamítnutí přístupu nepřihlášeným uživatelům.

[5] Prázdná hodnota.

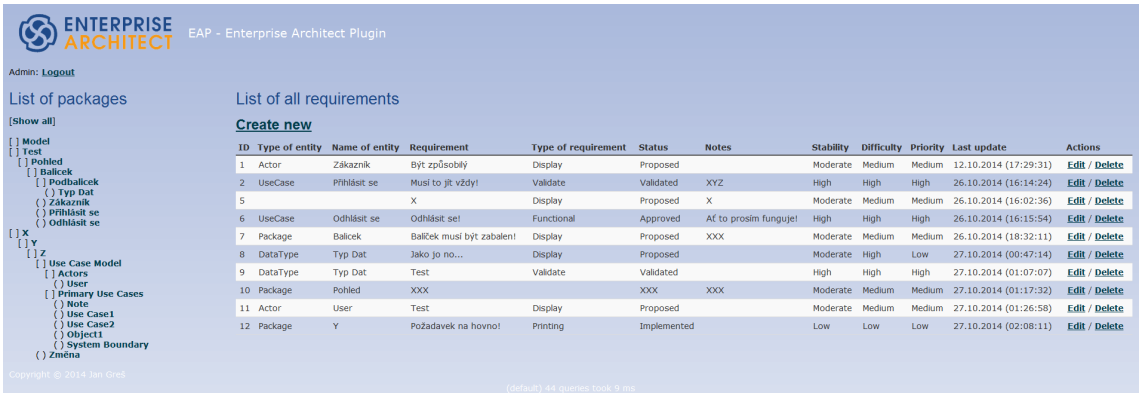
5.4.4 Změna layoutu

V této iteraci se celá aplikace poprvé dočkala nějakého layoutu a základního stylování. Design se snaží kopírovat původní prostředí EA a proto byly použity i jeho modrobílé barvy a pozadí s tímto gradientem. Vypůjčeno bylo i jeho logo.

Rozložení je pak standardní, konzervativní, se kterým by většina uživatelů počítačů neměla mít žádné větší problémy. Aplikace je v podstatě okenní tabulka, kde horní řádek obsahuje prostor pro logo, následuje tenké horizontální menu, dále je aplikace rozdělena do dvou velkých bloků (vlevo stromová struktura a vpravo výpis obsahu) a celé je to zakončeno horizontální patičkou.

V debug módu pak ještě následuje četná sekce vypisující všechny generované SQL požadavky směřující na databázi.

Pro lepší představu následuje ilustrační obrázek:



Obrázek 5.2: Rozložení layoutu 3. iterace

5.5 Zhodnocení

Výsledkem třetí iterace je nejenom prototyp, ale především už první samostatně funkční aplikace mající nějakou užitečnou funkcionalitu. Byla přidána autentizace a aplikovány role uživatelů, takže aplikaci je možné už někde proprietárně nasadit. Kompletního přepracování se dočkal její layout, který naznačuje, jakým grafickým směrem se bude vývoj dále ubírat. Ke slovu se dostaly také první grafické prvky, takže je aplikace už i trochu pohledná. A splněn byl i poslední úkol - implementace stromové struktury, což vneslo do celého projektu první známky přehlednosti.

Kapitola 6

4. iterace

6.1 Zadání iterace

Čtvrtá iterace má za úkol reagovat na zpětnou vazbu od zadavatele, zanalyzovat nové požadavky a aktualizovat dokumentaci. Dále se nese hlavně v duchu grafického šperkování zaměřeného především na líbivém vzhledu inspirovaném prostředím EA. V poslední řadě by zde měla vzniknout analýza dalších souvisejících elementů a jejich prvotní modelování.

6.2 Analýza

6.2.1 Sběr požadavků

Po konzultaci se zadavatelem byla v této fázi vývoje zjištěna jedna zásadní odchylka od jeho představ, způsobená zřejmě volnějším zadáním. Došlo zde totiž k záměně pojmů klasického *požadavku* a *požadavku na objekt*, kde se dosud vývoj aplikace zaměřoval především na mnohem obecnější *požadavek na objekt*. Tuto situaci je tedy třeba nějakým způsobem vyřešit.

Po prozkoumání databázových tabulek bylo zjištěno, že obyčejný *požadavek* je speciálním typem objektu (na který se v různých jiných případech mohou vázat *požadavky na objekt*). Protože je však entita *objekt* univerzální, obsahuje velice mnoho generických parametrů a atributů. Většina z nich se však pro potřeby typu *požadavek* vůbec nepoužívá a byly použity jejich výchozí hodnoty. Komplikace zde mohou vzniknout snad jen u atributu *ea_guid*, který je nějakým typem GUID [17] identifikátoru a který zatím není jak nasimulovat. Tento atribut se tedy dočasně ponechá prázdný a efekt na chování prostředí EA bude nadále sledován. Používané atributy se pak ve veliké většině překrývají s již modelovanou třídou požadavku. Některé však chybí a některé jsou zde navíc!

6.2.2 Analýza grafických prvků

Bylo by velice dobré, aby ku, spokojenosti uživatelů a co nejlepší intuitivnosti, aplikace měla podobné, nejlépe stejné, ovládací prvky jako mateřské prostředí EA. Ideální by pochopi-

telně byla nějaká jejich pohodlná extrakce, ale po zevrubném průzkumu souborové struktury aplikace EA nebyly nalezeny žádná volně dostupná grafická data.

Po načtení a exportu běžných zdrojů (resources) ze všech binárních souborů a exekutiv byla získána spousta ikon, kurzorů, bitmap a mnoho dalšího materiálu, ale bohužel zde nebylo to, co bychom potřebovali. Potřebné ikony jsou pravděpodobně uloženy jako PNG [24] obrázky a ač jsou sice uloženy v běžně dostupných zdrojích, jsou v binární podobě, přičemž jsem nenalezl nástroj, který by je dovolil „za letu“ prohlížet. Sice jsem si jich několik vyexportoval a složil ručně, ale jejich počet je natolik vysoký (řádově stovky, tisíce), aby bylo možné touto metodou najít, co bychom potřebovali. Tvorba externí utility by rovněž neuspořila mnoho času. Tudy cesta tedy také nevede.

Další metodou pak bylo ruční nafocení ikon přímo z aplikace za chodu, které bylo již i započato, ale nakonec také neuspělo pro jejich vysoký počet a přílišnou složitost mapování k objektům.

Rozhodl jsem se tedy zanalyzovat databázi, zda neobsahuje výčet všech možných typů entit v prostředí EA používaných. A obsahuje. Nalezneme jí v tabulce *t_objecttypes* a dokonce jsou zde metadata používaná k jejich mapování uživatelskou bitmapu! Toto je přesně, co potřebujeme. Prostředí EA totiž podporuje customizaci všech těchto ikonek implementovanou skrze jedinou bitmapu *UserImages.bmp* umístěnou v kořenovém adresáři EA. Jedná se o jediný soubor se všemi ikonami, na které jsou v databázi uloženy offsety. Pokrytí ikonek z EA tak bude dokonce bez výjimky 100

Obrázek 6.1: Bitmapa s uživatelskými ikonami

6.2.3 Analýza souvisejících elementů

Tato analýza byla z důvodu aplikace nutných změn přesunuta do následující iterace, kam bude mnohem lépe zapadat i logicky, protože je tam rovnou bude čekat i implementace.

6.3 Návrh

6.3.1 Návrh grafických prvků

Pro implementaci grafických prvků bylo navrženo nalezenou mapu postprocesovat a použít jako jediný obrazový soubor, který se bude v * * elementu posouvat pomocí invertovaného offsetu získaného z databáze. K tomuto účelu byl navrhnut nový model.

EntityIcon

Tento model mapuje tabulku *t_objecttypes* na již existující model *Entity* pomocí sdruženého sloupce *Object_Type* a slouží k získání nejenom offsetu pro grafická data, ale i názvu a popisu příslušného typu.

6.3.2 Úprava správy požadavků

Za současného stádia vývoje bylo rozhodnuto ponechat původní funkcionalitu správy *požadavků na objekt* a správu obyčejných *požadavků* k ní pouze přidat takovým způsobem, aby vše vypadalo velice homogenně, avšak uživatel netápal který požadavek je který.

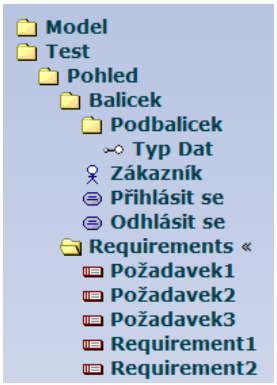
6.4 Implementace

6.4.1 Implementace grafických prvků

Grafická data byla nejprve zbavena šedivého pozadí a poté uložena jako 256 barevný transparentní GIF [16] soubor.

Pro snadné použití byla implementována globální funkce *EA_EntityIcon*, která po předání objektu *EntityIcon* vyrobí příslušnou grafickou ikonu díky správně posunutému offsetu. Framework CakePHP pro globální funkce nabízí skvělé umístění v souboru *Config/bootstrap.php*. Tuto funkci tedy můžeme použít ve všech view, ale i kdekoliv jinde. Je globální.

Stromová struktura tak byla doplněna o krásné ikony a dokonce bylo aplikováno i rozlišení otevřených složek a vybraných objektů. Pro otevřené složky byla natvrdo vybrána příslušná ikona z dostupné palety a všechny entity pak získaly označení v podobě francouzských uvozovek jako symbol vybrání.



Obrázek 6.2: Náhled stromové struktury

Do výpisů požadavků otevřených balíčků (a nikoliv vybraných objektů) pak byly přidány i obyčejné požadavky, díky čemuž musel vzniknout zcela nový controller, protože ač jsou oba požadavky velice podobné, tyto nové jsou objektem a je tedy proto nutné jejich implementaci zcela oddělit.

EntityController

Na tento controller se dostáváme z hlavního výpisu požadavků a jeho účelem je dosáhnout implementací stejného výsledku, jako jeho předešlý *RequirementController*, ze kterého vychází. Jejich API [12] je tak identické.

Máme tu funkci *edit(entity)* pro tvorbu nového požadavku s parametrem rovnajícím se nule, či úpravu již existující entity požadavku s *Object_ ID* v parametru. Dále je tu funkce *delete(entity)* přebírající totéž pro mazání a nechybí ani funkce *isAuthorised(user)* pro ochranu před neautorizovanými akcemi. Chybí pouze funkce *index(package , object)*, protože akci listování požadavky plní *RequirementController* pro oba typy požadavků.

Jako další nová funkcionalita jak do nového *EntityControlleru*, tak do původního *RequirementControlleru* bylo implementováno navracení do správné (dříve otevřené) složky/vybraného objektu po úspěšné manipulaci s požadavkem.

6.4.2 Drobné CSS úpravy

Kromě nových ikon aplikace doznala i drobných úprav co se grafických stylů týče. Byly odstraněny všechny nadbytečné fonty pozůstalé z proprietárního projektu CakePHP, dočkaly jsme se lepšího vzhledu stromové struktury a lépe vypadají i formulářové prvky.

6.5 Testování

6.5.1 EntityController

Protože tento controller je zcela nový, bylo nutné otestovat všechny jeho funkce. Úprava a mazání probíhají v naprostém pořádku, ale narazil jsem na zásadní chybu při přidávání nového požadavku. Chyba se týká již dříve zmiňovaného GUID, který zřejmě plní svou funkci přesně dle své podstaty. Databáze má na tomto sloupci unikátní klíč a tedy nám trik s prázdným GUID neprošel. Rozhodl jsem se toto obejít MD5 hashí časového razítka, což prozatím zajistí požadavek na unikátnost.

6.6 Zhodnocení

Výsledkem čtvrté iterace je už vcelku vyspělá aplikace splňující značnou část svého zadání. Narazili jsme na četné problémy s nejasným zadáním, ale úspěšně jsme se s nimi vypořádali. Grafická stránka byla vylepšena o nové prvky a aplikace ikon z prostředí EA byla zvládnuta naprosto skvěle. Podařilo se vypilovat i dokumentaci, která je nyní na velice slušné úrovni.

Nestihla se pouze analýza souvisejících elementů, ale ta by měla bez větších potíží být vyřešena v následující iteraci, po dalších připomínkách klienta.

Kapitola 7

5. iterace

7.1 Zadání iterace

Pátá iterace má za úkol nejprve provést analýzu souvisejících elementů požadavku, které se nestihly koncem předešlé čtvrté iterace a déle se jimi zabývat. To znamená navrhnout nějaké technické provedení a to poté je i implementovat a otestovat. Celá tato iterace tedy bude patřit hlavně těmto souvisejícím elementům.

7.2 Analýza

7.2.1 Co jsou to související elementy?

Ještě než začneme se samotnou analýzou je potřeba dát sousloví *související elementy* nějaký konkrétní smysl. Co se tím vlastně myslí? Ze zadání to není přesně patrné, ale po konzultaci s klientem byla předložena představa, že se jedná o širší kontext daného požadavku, který je zároveň entitou.

Tyto typy požadavků pak na sebe mohou navazovat různé další související elementy. Z pohledu uživatele kupříkladu diagram, ve kterém jsou obsaženy, sousedící entity a především vazby, včetně jejich typů, kterými jsou vzájemně propojeny. Zkrátka a dobře aby člověk užívající aplikaci EAP měl širší pojem o tom, s čím to vlastně manipuluje a jaký to bude mít dopad na zbytek právě otevřeného projektu.

7.2.2 Analýza souvisejících elementů v databázi

Jakmile už víme co zhruba hledáme, můžeme se to pokusit najít ve strukturálně definované databázi prostředí EA. Středobodem se zdá býti tabulka *t_diagram*, která uchovává informace o jednotlivých diagramech. K té se pak dají vyhledat zúčastněné entity z tabulky *t_object*, kterou již definovanou jako třídu máme. Mapovací tabulka se příhodně jmenuje *t_diagramobjects*. Tím jsme dokázali přidružit jednotlivé entity do svých diagramů. Co ale s vazbami?

Vazby jsou uchovávány ve speciální tabulce *t_ connector* a opět jsou plně mapovatelné do příslušných diagramů, tentokrát tabulkou *t_ diagramlinks*. Co je ale ještě mnohem více přínosnější je, že jednotlivé vazby mohou být mapovány přímo na entity, které vzájemně spojují. Tedy na tu počáteční i konečnou.

7.2.3 Analýza jednotlivých tabulek

Každá ze čtyř nově využívaných tabulek obsahuje nespočet sloupců nesoucích hromadu informací. Nyní je potřeba z nich vybrat ty, které bychom mohli pro projekt využít.

t_diagram

Spíše menší tabulka, obsahující hlavně jméno a typ diagramu. Také z ní lze určit umístění diagramu v balíčku, jeho autora a verzi. Ostatní sloupečky se zdají být nepoužitelné. Zkoušel jsem ještě využít rozměry diagramu, ale ty se z neznámého důvodu po jeho zvětšení v prostředí EA nemění. Jsou stále fixní.

t_diagramobjects

Spojovací tabulka pro diagramy a entity. Kromě cizích klíčů zde ale najdeme velice cenné údaje o umístění jednotlivých entit v diagramu. Jsou kde k dispozici souřadnice X i Y jak levého horního, tak pravého dolního bodu každé obdélníkové entity. Po odečtení těchto bodů lze velice snadno získat i šířku a délku. Jedinou záludností pak může být fakt, že se celý diagram nachází ve čtvrtém kvadrantu kartézské soustavy souřadnic. Vertikální souřadnice jsou tedy všechny záporné. Nacházíme zde i souřadnici Z pro určení překryvu jednotlivých entit, ale ta je spíše jakýmsi pořadím, než plnohodnotnou souřadnicí. Entity jsou číslovány od jedné vzestupně, kde číslo jedna má vždy entita v nejvyšší vrstvě, tedy překrývající všechny ostatní.

t_diagramlinks

Spojovací tabulka pro diagram a vazby. Kromě cizích klíčů jsem zde nenašel nic, co by se dalo nějak využít. Jsou zde sice nějaké souřadnice v podobě metadat, ale jejich transformaci jsme neodhalil.

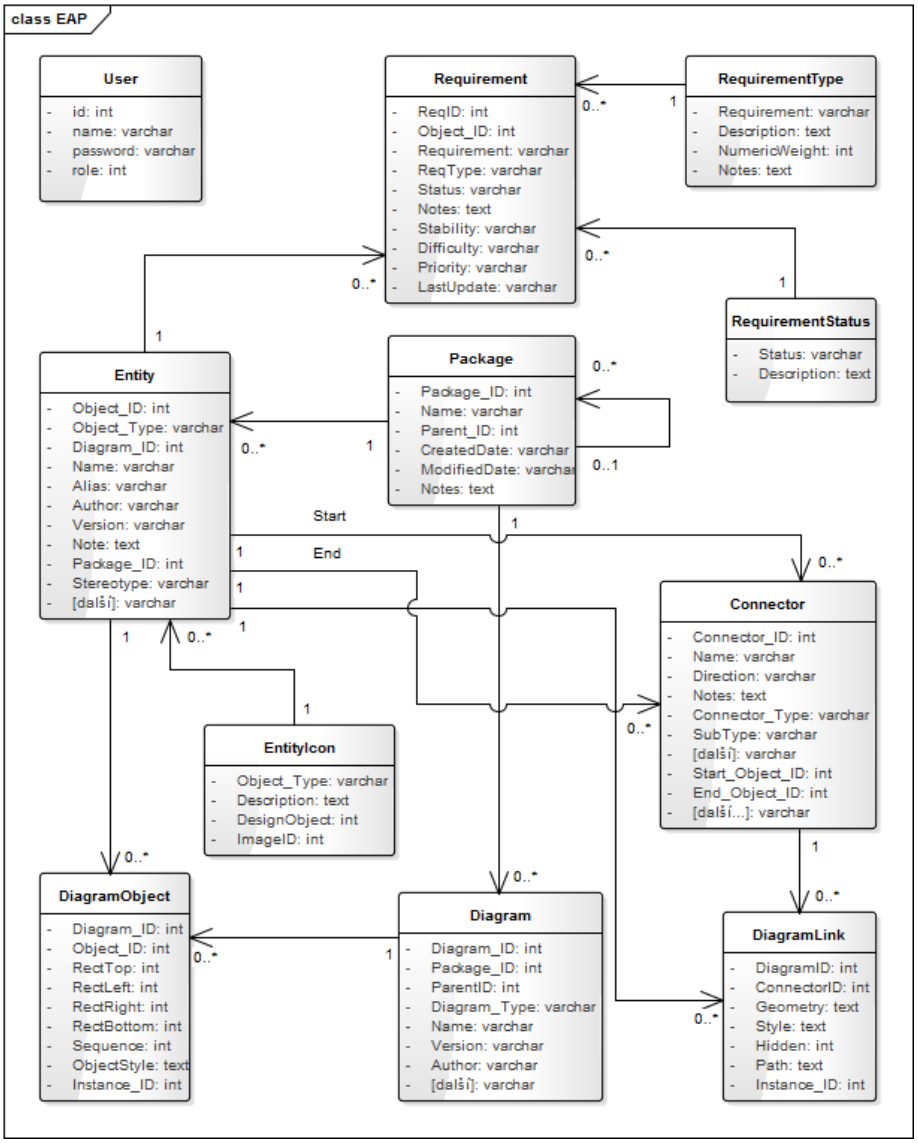
t_connector

Tabulka reprezentující jednotlivé vazby a nutno dodat, že jedna z největší vůbec. Vazby se sice dají pojmenovávat, ale to se využívá spíše zřídkakdy, proto pro nás bude důležitý zejména typ a směr jednotlivých vazeb. Výjimečně pak ještě podtyp využitelný například u odlišení agregace a kompozice. Nejdůležitější pak tedy budou cizí klíče pro počáteční a koncové vzájemně spojené entity.

7.3 Návrh

7.3.1 Návrh nových tříd

Z analýzy získaných tabulek nyní vytvoříme nové třídy, které se stanou dalšími modely v našem projektu EAP. Nejprve si jednotlivé modely popíšeme a nakonec nebude chybět ani obrázek s diagramem tříd 7.1.



Obrázek 7.1: Diagram tříd 5. iterace

Diagram

Model představující diagram bude využit hlavně k zobrazení informací o vybraném diagramu. Je reprezentován tabulkou *t_ diagram* a může mít množinu tříd *DiagramObject* a *DiagramLink* obě napojené přes *Diagram_ ID*. Tyto vztahy jsou realizovány pomocí vazby *hasMany*. Samotný model pak obsahuje i *Package_ ID*, pomocí kterého je napojen na třídu *Package* vazbou *belongsTo*.

DiagramObject

Model poskytující hlavně souřadnice jednotlivých entit v diagramu. Lze pomocí něho také určit, ke kterému diagramu právě otevřený požadavek zrovna náleží. Je reprezentován tabulkou *t_ diagramobjects* a pomocí vazby *belongsTo* je napojen na třídy *Diagram* a *Entity*.

DiagramLink

Model reprezentující vazbu M...N mezi třídou *Diagram* a *Connector*. Sám o sobě nenese žádné podstatné informace. Je reprezentován tabulkou *t_ diagramlinks*. Ve vazbě *belongsTo* nalezneme tedy pouze třídu *Connector*.

Connector

Model představující samotné vazby mezi entitami. Obsahuje všechny potřebné informace a atributy o tomto spojení. Je reprezentován tabulkou *t_ connector* a pomocí vazby *belongsTo* je hned dvakrát napojen na třídu *Entity* a to pomocí *Start_ Object_ ID* a *End_ Object_ ID*, které představují počínající a koncovou entitu, jež se účastní vazebného vztahu.

7.3.2 Návrh grafického zobrazení

Po namodelování výše uvedených tříd je potřeba je nějak využít. Celé jejich využití se bude vztahovat pouze na otevřenou entitu *Entity*, tedy obyčejný požadavek obsluhovaný controllerem *EntityController*, který byl implementován v minulé iteraci.

Pod možností uložit provedené změny budou k dispozici tři informační tabulky. Nejprve informace o diagramu, ve kterém je prohlížený požadavek obsažen, dále všechny jeho sesterské entity (tedy nejenom požadavky, ale i libovolné jiné entity ve stejném diagramu) a nakonec seznam všech vazeb v tomto diagramu obsažených včetně jejich počátečních i koncových entit.

Pro lepší přehlednost nebude chybět ani názorný diagram, který bude implementován svépomocí, díky přesným souřadnicím, získaným z třídy *DiagramObject*. Tento diagram bude implementován pomocí tagu *canvas* standardu HTML5. Diagram bude navržen jako statický. A vůbec všechny související elementy jsou určeny pouze pro čtení, aby byl dodržen informační kontext.

7.4 Implementace

7.4.1 EntityController

Zde došlo pouze k drobným úpravám v metodě *edit(entity)*, kde je potřeba na konci z připravených modelů načíst všechna doplňující metadata. Postupně tak načteme a do příslušného view přes privátní metodu *set* propagujeme proměnné *diagram* a *diagram_ data*.

Ta první představuje mateřský diagram, který načítáme skrze model *DiagramObject*, protože právě ten obsahuje jak cizí klíč k diagramům, tak k jednotlivým entitám, včetně té naší. Ale tak jak máme třídy namodelovány model *DiagramObject* použít nemůžeme. Je zapotřebí použít metodu *unbindModel* a odstranit vazbou *belongsTo* napojený model *Entity*. Kdybychom to neudělali, tak zbytečně cyklicky načítáme redudantní data a dostaneme se zpátky k naší entitě. Poté už můžeme pomocí *Object_ ID* pohodlně zjistit ID mateřského diagramu.

Pomocí získaného *Diagram_ ID* pak již načteme kompletní strukturu diagramu včetně množin tříd *DiagramObject* s napojenými entitami a *DiagramLinks* s napojenými vazbami. To je v controlleru vše. Zbytek už bude práce view a HTML5. Pro úplnost ještě úryvek zdrojového kódu:

```
$DiagramObject = ClassRegistry::init('DiagramObject');
$DiagramObject->unbindModel(Array('belongsTo' => Array('Entity')));
$diagram = $DiagramObject->find('first' , Array(
    'conditions' => Array(
        'Object_ID' => $id
    ),
    'recursive' => 2
));
$this->set('diagram' , $diagram);

if (!Empty($diagram)) {
    $Diagram = ClassRegistry::init('Diagram');
    $diagram_data = $Diagram->find('first' , Array(
        'conditions' => Array(
            'Diagram_ID' => $diagram['Diagram']['Diagram_ID']
        ),
        'recursive' => 3
    ));
}
else {
    $diagram_data = Array('DiagramObject' => Array() , 'DiagramLink' => Array());
}
$this->set('diagram_data' , $diagram_data);
```

7.4.2 edit.ctp

V tomto view se všechna nově načtená data zobrazí do příslušných tabulek. Tabulka s informacemi o mateřském diagramu (zobrazující jméno, typ, jméno balíčku, autora a verzi), s výčtem sesterských (zobrazující jméno a typ) a s výčtem vazeb v diagramu (zobrazující typ, směr a počáteční i koncovou entitu) jsou zobrazeny přímo v tomto view, zatímco diagram je delegován do externího elementu s názvem *diagram.ctp*.

7.4.3 HTML5 diagram

Načtená data obsahují v polích množinu entit i vazeb. Ty je nejprve nutné převést z PHP do JavaScriptových objektů následující struktury.

Objekt Object

```
:Object {
    left: int
    top: int
    width: int
    height: int
    type: string
    title: string
    selected: boolean
}
```

Objekt Line

```
:Line {
    startX: int
    startY: int
    endX: int
    endY: int
    type: int
}
```

Objekt *Object* je prakticky jen výtahem dat s minimálním postprocessingem. Záporné hodnoty souřadnic kartézské soustavy jsou ošetřeny absolutní hodnotou a rozměry jako šířka a výška jsou výsledkem odečtení levého horního od pravého dolního rohu. Tyto hodnoty vycházejí z třídy *DiagramObject*. Typ a název pak vycházejí ze třídy *Entity*. Booleanovská hodnota *selected* pak pouze říká, zda je daný objekt tím, který máme právě otevřený. Zde se porovnají *Object_ ID*.

Objekt *Line* je však už ale složitější. Je zde přítomen algoritmus určující, kde bude vazba začínat a kde končit. Protože jsou všechny objekty obdélníky, nabízí se pouze čtyři možnosti, kde by to mohlo být. Jsou jimi poloviny všech čtyř stran. Algoritmus má tak na starosti zajistit, aby byly vazby co nejpřirozenější a začínaly i končily na sobě nejbližších stranách. Objekty jako takové jsou totiž po canvasu rozmístěny fixně dle přesně zadaných souřadnic o přesně zadaných rozměrech. K vazbám však tyto souřadnice nemáme a musíme si tedy poradit sami.

Je-li počáteční objekt (A) kdekoliv plně nad koncovým objektem (B), povede vazba z prostředku spodní hrany A do prostředka horní hrany B. Je-li objekt A kdekoliv plně pod objektem B, povede vazba z prostředku horní hrany A do prostředka spodní hrany B. Jsou-li ovšem objekty A a B nějakým způsobem ve vertikální kolizi, ptáme se, zda je objekt A plně vlevo od objektu B (pak vede vazba z prostředka pravé hrany objektu A do prostředka levé hrany objektu B) nebo zda je objekt A plně vpravo od objektu B (pak vazba vede z prostředka levé hrany objektu A do prostředka pravé hrany objektu B).

Poslední atribut typ pak určí, zda bude mít vazba šipku na svém počátku, na svém konci, na obou nebo zda nebude mít šipku žádnou.

Správně zpracovaná data se pak již jen vykreslí do předpřipraveného canvasu. Jeho velikost se určí ještě před vykreslením a to tak, aby se nacházela těsně za nejvzdálenějšími objekty jako směrem vpravo, tak směrem dolu. Objekty jsou zelené ve stylu prostředí EA s černým okrajem, který je u právě prohlíženého objektu zesílen. Hrany jsou zatím pouze ostré. Všechny objekty sice vypadají identicky bez jakékoliv grafiky, ale jako mockup diagramu to bohatě postačuje. Uprostřed každého objektu je navíc informace o jeho typu i názvu, takže uživatel nemůže tápat.

Vazby jsou pak vykresleny pomocí prostých čar s plnou šipkou na konci, je-li nějaká potřeba. Odlišení jednotlivých vazeb zde zatím není implementováno. Pro zajímavost uvádím funkci pro HTML5 JavaScript, vykreslující šipku libovolné rotace s fixem pro čistě vertikální variantu:

```
function arrow(ctx , x1 , y1 , x2 , y2 , type) {
    if (type == 0) return;
    var x = 0;
    var y = 0;
    var radius = Math.atan((y2 - y1) / (x2 - x1));
    if (type == 1) {
        x = x1;
        y = y1;
        radius += ((x2 > x1) ? -90 : 90) * Math.PI / 180;
    }
    if (type == 2) {
        x = x2;
        y = y2;
        radius += ((x2 > x1) ? 90 : -90) * Math.PI / 180;
```

```
    }
    if (radius == 0) {
        radius = Math.PI;
    }
    else if (radius == Math.PI) {
        radius = 0;
    }
    ctx.save();
    ctx.beginPath();
    ctx.translate(x , y);
    ctx.rotate(radius);
    ctx.moveTo(0 , 0);
    ctx.lineTo(5 , 20);
    ctx.lineTo(-5 , 20);
    ctx.closePath();
    ctx.restore();
    ctx.fill();
}
```

7.5 Testování

7.5.1 HTML5 diagram

V této fázi vývoje byla největším testům podrobena grafika vykreslující diagram souvisejících objektů. Byly zkoušeny všechny možné bezkolizní pozice objektů a vazby vždy vypadají dobře a šipky mají správně umístěné. Jediná chyba byla nalezena v případě, že je vazba čistě vertikální (tedy se počáteční hodnota X nemění od koncové). V takovém případě je potřeba vyměnit nulu za π a vše již funguje jak má. Testy dále zjistily, že příliš dlouhý text z příliš úzkých objektů přesahuje mimo jeho hrany. Tato skutečnost je na zvážení zadavatele, zda jí ponechat nebo nějak dále řešit. Jinak je diagram plně funkční.

7.6 Zhodnocení

Výsledkem páté iterace je přesně to, co si dala za cíl. Přidaná podpora souvisejících elementů, které byly zvládnuty od jejich analýzy, přes modelování nově potřebných tříd až po jejich implementaci včetně grafického vykreslení. Opomenuta byla pouze manipulace s nimi, která bude přidána volitelně v některé z dalších iterací a je na zadavateli, zda toto bude vyžadovat či nikoliv. Zadavatel dále rozhodne, zda se dále věnovat grafickému pilování diagramu (podpora oblých rohů objektů, více druhů šipek - např. přerušovaných a nevyplněných, přidání symbolů kompozice a agregace apod.). Jinak proběhla 5. iterace velmi zdařile.

Kapitola 8

6. iterace

8.1 Zadání iterace

Šestá iterace měla původně za úkol do aplikace přinést verzování požadavků a umožnit i jejich grafický přesun. Po konzultaci se zadavatelem se ale vývoj celé aplikace přesunul zcela někam jinam. Přesouvání požadavků mezi balíčky je v aplikaci již přítomno z předešlých iterací v negrafické podobě a představa verzování zadavatele se nestřetla s představou realizátora. Po letmé analýze se ukázalo, že by její implementace vystačila na samostatný projekt a proto bylo rozhodnuto se ve vývoji zaměřit na multipoužitelnost celé aplikace pro více databázových instancí a umožnit uživatelům nezávislé registrace. Toto jsou tedy dva stěžejní body šesté iterace.

8.2 Analýza

8.2.1 Podpora více databázových instancí

Pod tímto pojmem se nemyslí nic jiného, než možnost měnit konfigurační nastavení s připojením k databázovému serveru nejenom globálně, ale i lokálně, pro každého uživatele. Představa je taková, aplikace poběží dedikovaně jako jediná instance na nějakém centrálním, rychlostně dobře dostupném, serveru a bude sloužit vícero souběžným účastníkům, kde každý může být připojen k úplně jinému serveru, s jiným uživatelským jménem, heslem a k jiné databázi. Z aplikace se tak najednou stává služba, což je v dnešní době modernější a žádanější přístup. Zadavatel se tak stává teoreticky jediným poskytovatelem.

Předmětem analýzy bylo hlavně zjistit, zda tento zdánlivě jednoduchý požadavek je vůbec v prostředí zvoleného frameworku CakePHP možno realizovat. První výsledky byly spíše negativní. Framework, jež je silně modelově založen, pracuje se statickým konfiguračním souborem *Config/database.php* obsahujícím konfigurační třídu *DATABASE_CONFIG*, která v sobě ukrývá jednotlivá pojmenovaná konfigurační pole použitelné pro interní PDO [3] spojení s databází. Ty jsou staticky přidruženy ke každému modelu veřejnou proměnnou *useDbConfig*.

Toto představovalo celkem veliký problém a padaly varianty, že bude muset být pro každou instanci předgenerován nový model a upraven i konfigurační soubor, ale to by bylo tak složité a nepraktické, že by snad bylo lepší tuto celou funkcionalitu oželet.

Částečné řešení však přinesla až komunita uživatelů na serveru *StackOverflow*[1], kde bylo popsána dynamická změna databázové konfigurace modelu za běhu, avšak pouze výběrem jiného předpřipraveného nastavení v souboru *Config/database.php*. Důležité však je, že jde alespoň toto.

8.2.2 Registrace a správa uživatelů

Aby byla transformace aplikace do podoby služby završena kompletně, navrhl zadavatel implementaci nějaké centrální správy uživatelských instancí a účtů, popřípadě možnost nezávislé registrace. Nakonec byl učiněn kompromis a byl zvolen model registrace administrátorů, kteří si dále již vytvoří sami své vlastní poduživatele, které včetně sebe samého již samostatně nakonfigurují. Tato varianta by měla být do produkčního nasazení asi nejpraktičtější.

8.3 Návrh

8.3.1 Řazení dat v modelech

U několika klíčových modelů (*Requirement*, *Package* a *Entity*) byla do konfigurace přidána klíčová proměnná *order*, čímž se zajistilo výchozí řazení vybíraných dat pomocí funkce *find*. V praxi šlo o abecední řazení vypisovaných požadavků, entit i balíčků. Žádné nové modely v této iteraci již nebyly přidány.

8.3.2 AppModel a dynamický výběr databázové konfigurace

Framework CakePHP je silně objektově orientovaný. Všechny deklarované modely jsou třídy rozšiřující bázovou třídu *AppModel*, která je přítomna v každé aplikaci a která byla dosud ponechána prázdná, bez povšimnutí.

Nyní je zde překryta veřejná funkce *getDataSource()* (deklarovaná v abstraktní třídě *Model* vracející objekt databázového zdroje, na kterém jsou vykonávány všechny sestavené SQL [26] dotazy. Nyní je tento objekt, na základě rady komunitních přispěvatelů, vracen dynamicky, což je realizováno včasnou změnou konfigurace. Výsledkem však mohou být pouze dva pojmenované zdroje dat:

1. *default* - Výchozí databázová konfigurace, která je jako jediná fyzicky uložena v konfiguračním souboru *Config/database.php*. Slouží pouze pro jediný lokální model *User*.
2. *ds* - Dynamická databázová konfigurace, která není nikde fyzicky uložena a vzniká v *AppControlleru* při každém jeho zpracování v závislosti na právě přihlášeném uživateli.

[1] <http://stackoverflow.com/>

Jak je tedy z výše uvedeného asi patrné, databáze uživatelů s tabulkou *e_ users* byla zcela oddělena od jakýchkoliv jiných provozních dat prostředí EA. Ty jsou nyní načítány z vlastních databázových poskytovatelů.

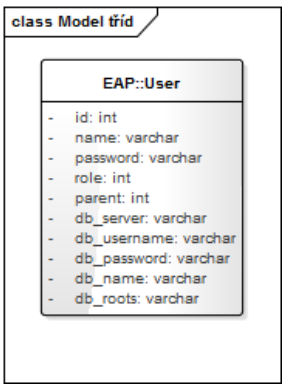
Pro úplnost výpis samotného *AppModelu*.

```
<?php
    App::uses('Model' , 'Model');

    class AppModel extends Model {
        public function getDataSource() {
            $source = $this->alias == "User" ? "default" : "ds";
            if ($source !== null && $source !== $this->useDbConfig) {
                $this->setDataSource($source);
            }
            return parent::getDataSource();
        }
    }
?>
```

8.3.3 Registrace

Původní verze aplikace počítala s výchozími uživatelskými účty, které se při prvním úspěšném spojení s mateřskou databází společně s tabulkou *e_ users* instalovaly automaticky. Tomu tak již není. Instaluje se pouze tabulka uživatelů a to v nové podobě.



Obrázek 8.1: Diagram uživatelské třídy

Byly přidány následující nové sloupce:

parent

Obsahuje ID mateřského uživatele. V případě bázového účtu je zde nula.

db_server

Obsahuje jméno lokálního či vzdáleného serveru. Může zde být jak IP adresa [19], tak DNS [15] název a to včetně správného portu.

db_username

Obsahuje přihlašovací jméno k databázi.

db_password

Obsahuje přihlašovací heslo k databázi v otevřeném *plaintextovém* tvaru.

db_name

Obsahuje jméno databáze, ke které se má systém připojit a která obsahuje platnou instanci projektu prostředí EAP.

db_roots

Obsahuje seznam kořenů načtených v levé stromové struktuře.

Čerstvě nainstalovaný systém EAP je tedy nyní zcela prázdný a jednotliví uživatelé se musí sami nejprve zaregistrovat a vytvořit si svůj účet. Registrace není ničím omezena ani nikým schvalována, dokonce nevyžaduje ani ověření pomocí platné e-mailové adresy. Účet si tak z hlavní stránky může velice jednoduše vytvořit naprosto každý a po jeho tvorbě se ihned přihlásit a začít aplikaci plně používat.

8.3.4 Nové uživatelské role

S novým uživatelským pojetím došlo i k mírnému přepracování uživatelských rolí. Základní trojice zůstává, avšak každá role má trochu jiné možnosti než dříve.

Administrátor (Admin)

Při samostatné registraci vzniká účet s nejvyššími oprávněními - *Admin*. Tento účet má jako jediný možnost vytvářet (upravovat i mazat) v nastavení profilu další uživatele libovolných tří rolí, kteří jsou mu plně podřízeni. Může jim tedy kdykoliv změnit jméno, heslo, ale především jim musí nastavit konfigurační data, tedy ke kterému serveru se s jakým přihlašovacím jménem a heslem mohou připojit a k jaké databázi. Také jim může nastavit omezení v podobě přístupu do pouze některých větví stromové struktury balíčků prostředí EAP. Všechny tyto hodnoty a nastavení běžní uživatelé nemohou měnit a ani je nevidí. *Admin* je pochopitelně nastavuje i sobě. Jemu nepodřízené uživatele pochopitelně spravovat nemůže, takže jednotliví *Admini* pracují zcela odděleně.

Uživatel (User)

Běžný účet, který může plně pracovat ve své přidělené instanci. Nově si může v nastavení profilu změnit jméno i heslo. Nemůže vytvářet žádné další poduživatele, ani spravovat své připojení a kořenové balíčky.

Host (Guest)

Omezený účet, který je určen pouze pro čtení (prohlížení) své přidělené instance. Účet nemůže v databázi cokoliv změnit a to ani své jméno či heslo. Vůbec nemá přístup do nastavení profilu. Nově může libovolné entity alespoň otvírat, avšak nemá k dispozici tlačítko pro ukládání provedených změn.

8.4 Implementace

8.4.1 AppController

Nejdůležitějších implementačních změn se dočkal *AppController*, tedy bázovou třídou pro všechny ostatní controllery. Ten nyní využívá statickou třídu *ConnectionManager* k dynamickému vytváření databázových konfigurací za běhu, které se pak ukládají pod již zmíněným jménem *ds*. Tato implementační část je již mou vlastní invencí rozšiřující nápad komunity vývojářů frameworku CakePHP na dynamické přepínání databázových konfigurací u modelů za běhu.

Vše se odehrává v metodě *beforeFilter()*, která se volá u každého controlleru ještě před tvorby instancí jeho modelů, což je velmi důležité. Nejprve se pomocí čistého SQL zeptáme na přítomnost tabulky uživatelů, kterou v případě absence automaticky doinstalujeme. Dále se pak ověří a načtou uživatelská data a pomocí *ConnectionManageru* dynamicky vytvoří nová databázová konfigurace na základě načtených přihlašovacích údajů. Pojmenována je pochopitelně *ds* tak, aby jí každý model správně našel.

Tento úsek kódu si zde opět dovolím citovat, protože je v implementaci multipoužitelnosti zcela klíčový:

```
(...)
    $ds = ConnectionManager::getDataSource("default");
    $check = $ds->query("SHOW TABLES LIKE 'e_users'");
    if (Count($check) == 0) {
        $ds->query(Implode("" , File(ROOT."/".APP_DIR."/Install/e_users.sql")));
        $this->Session->write('first' , true);
        $this->redirect(Array('controller' => 'users' , 'action' => 'login'));
        //Header("Location: /".APP_DIR);
        Die();
    }

    $this->user = $this->Auth->user();
    $this->set('user' , $this->user);
```

```
    if ($this->user != null) {
        $config = Array(
            'datasource' => 'Database/Mysql',
            'persistent' => false,
            'host' => $this->user['db_server'],
            'login' => $this->user['db_username'],
            'password' => $this->user['db_password'],
            'database' => $this->user['db_name'],
            'prefix' => '',
            'encoding' => 'utf8'
        );
        $connection = null;
        try {
            $connection = ConnectionManager::create('ds' , $config);
        }
        catch (Exception $e) {
            ConnectionManager::drop('ds');
        }
        $this->connected = $connection != null;
        $this->set('connected' , $this->connected);
(...)
```

Jako další vylepšení *AppControlleru* pak byly přidány různé podmíněné načítání dat, aby systém nejenom nenačítal nic zbytečně, ale hlavně aby se zabránilo různým chybám a výjimkám v případě špatně konfigurovaného připojení. Systém nově tuto skutečnosti sám rozpozná a oznámí v horním menu. Pokud není připojení k databázi dostupné, rovněž automaticky načte profilové nastavení s možností změny přístupových údajů.

Poslední invencí je pak dynamické načítání stromové struktury balíčků prostředí EA z předdefinovaných kořenů. To umožňuje definovat nejenom hlouběji zanořené větve jako kořenové, ale především definovat i vícero kořenových větví najednou, které se mohou dokonce i různě překrývat.

Zde však vyvstávají dva problémy k budoucímu zvážení a řešení. Jedním je potřeba implementace postupného načítání jednotlivých větví, které se momentálně načítají jako celistvý rozbalený strom s omezením v hloubce pět. Toto bude předmětem sedmé iterace. Druhý problém je pak bezpečnost. Důležité je zde připomenout, že přidělení pouze některých větví podřízenému uživateli má pouze přehledový charakter, nikoliv bezpečnostní. Systém pracuje vždy s úplnou množinou všech dostupných balíčků a tyto nabízí i v módech editace. Prakticky je tedy možné přesunout entitu do balíčku, který vůbec nevidíme. Dosazením správného parametru do URL [27] adresy pak můžeme dokonce tyto jinak neviditelné entity načíst a dále modifikovat. Bylo by samozřejmě dobré toto vyřešit, ale zatím je to odloženo z důvodu přílišné složitosti ověřování oprávnění, kdy by se muselo při každé operaci traverzovat od kýžené entity až k platně definovanému kořeni či se došlo na konec a přístup byl zamítnut. Toto by velice zpomalovalo chod celé aplikace a vznikly by stejné problémy s limitem hloubky.

8.4.2 EntityController

Zde nebyly provedeny žádné větší změny kromě úpravy autorizace oprávnění. Role *Uživatel* nyní je autorizován k akci *edit(id)*, ale nemůže do ní posílat žádná změnová data.

8.4.3 RequirementController

Zde nebyly provedeny žádné větší změny kromě úpravy autorizace oprávnění. Role *Uživatel* nyní je autorizován k akci *edit(id)*, ale nemůže do ní posílat žádná změnová data.

8.4.4 UsersController

V tomto controlleru došlo k výrazným změnám. Přibyla metoda *beforeFilter()* definující akci *register()* jako veřejnou a samozřejmě tato akce samotná v podobě metody. Její úlohou je prostě vytvořit nového uživatele se zadaným jménem a heslem a výchozími ostatními hodnotami.

Přibyla pak zcela nová metoda *profile(id)*, která v sobě ukrývá kompletní logiku správy uživatelských profilů. Pokud jí voláme jako metodu, načítá data. Pokud do ní data posíláme, ukládá je. Důležitý je parametr *id*, který ovlivňuje chování hodně.

- *id = null*: metoda pracuje s naším vlastním profilem
- *id = 0*: metoda pracuje s virtuálním novým profilem, který je schopna vytvořit
- *id = [kladné číslo]*: metoda pracuje s profilem daného *id*, který je schopna uložit
- *id = [záporné číslo]*: metoda pracuje s profilem daného *id* v absolutní hodnotě, který je schopna smazat

Pokud je zasílaných datech přítomno nové heslo a není-li prázdné, zahashuje se a uloží. Toto je výhodné pro případ, kdy chceme změnit uživatelskou konfiguraci, ale heslo chceme ponechat beze změny. Změna jména se projeví okamžitě, zatímco na nové heslo budeme dotázáni až při příštím přihlášení.

Pokud se jedná o editaci vlastního profilu, je proměnná *parent* vždy vynucena na nulu, zatímco pokud pracujeme s cizím účtem, je vynucena na naše vlastní *id* profilu.

Na bezpečnost v tomto controlleru byl kladen maximální důraz. Nemělo by se tedy stát, že by někdo upravil cizí účet nebo provedl jinou neautorizovanou operaci. Pokud se nejedná o administrátorský účet, všechna vstupní data kromě *id*, *name* a *password* jsou účelově zahozena.

Co se týče autorizace samotné, tak pouhý *Host* do celé akce *profile(id)* vůbec nemá přístup. Běžný *Uživatel* pak má přístup jen a pouze bez použití parametru a nebo s parametrem rovným svému *id*. Bohužel i zde narážíme na limity autorizování akcí frameworku CakePHP a musíme si pomoci jinak. Aby nenastala hypotetická možnost, že *Administrátor* (který může

akci *profile(id)* využívat naplno) nezměnil data jemu nepodřízeného jiného uživatele, musely být kromě výchozích funkcí *Model->save([data])* a *Model->delete([primární klíč])* použité jejich obecnější alternativy umožňující větší bezpečnosti.

Funkce *Model->saveAll([data] , [nastavení])* pracuje s podobnou strukturou dat, jakou známe z funkce *Model->find([typ výběru] , [nastavení])*. Objevuje se zde tedy i klauzule *conditions*, kterou můžeme ukládání omezit. Následuje ukázka použití:

```
if ($this->User->saveAll($this->request->data , Array(
        'conditions' => Array(
            'id' => $this->User->id,
            'or' => Array(
                'id' => $this->user['id'],
                'parent' => $this->user['id']
            )
    )))) {
    $this->Session->write('Auth' , $this->User->read(null , $this->Auth->User('id')));
    $this->Session->setFlash(__('Profile was successfully saved.'));
    if ($id == null) {
        $this->redirect(Array('controller' => 'Requirement' , 'action' => 'index'));
    }
}
else {
    $this->Session->setFlash(__('Error during saving!'));
}
```

Tento úryvek kódu demonstruje jak uložení pouze v případě, že ukládáme buď data k svému vlastnímu profilu nebo k profilu nám podřízenému, tak aktualizaci objektu *Auth* v přihlášeném sezení.

Funkce *Model->deleteAll([podmínky])* pak funguje obdobně s rozdílem, že zde máme možnost už pouze vložit omezující podmínky. Nejlépe to ukážeme na příkladě:

```
if ($this->request->is('get')) {
    if ($this->user['role'] < 1 && $id < 0) {
        if ($this->User->deleteAll(Array(
                'id' => Abs($id),
                'parent' => $this->user['id']
            ))) {
            $this->Session->setFlash(__('Profile was successfully deleted.'));
            $this->redirect(Array('action' => 'profile'));
        }
    }
}
```

Tento úryvek kódu demonstruje jak omezení operace pouze pro *Administrátora*, tak samotné podmíněné mazání, kdy je možné mazat pouze sobě podřízené uživatele, nikoliv už však sebe!

Jedinou nevýhodou výše použitých funkcí je fakt, že i při pokusu uložit či smazat data nepodřízeného uživatele bude vypsána kladná stavová hláška. Operace však neproběhne!

8.4.5 menu.ctp

Zde přibyla stavová informace o úspěšném připojení k databázi. V případě kladného stavu je zde vidět název serveru, přihlášené jméno i databáze. Skryto je pouze heslo. Tuto informaci mají k dispozici všechny uživatelské role. Běžný *Uživatel* a *Administrátor* zde pak mají nově navíc ještě nabídku pro správu profilu.

8.4.6 edit.ctp

Šablona pro úpravu jak normálního požadavku, tak objektového doznala drobné úpravy v podobě absence tlačítka *Uložit* pro obyčejné *Hosty*.

8.4.7 index.ctp

V indexu požadavků bylo pouze z estetických důvodů změněno tlačítko *Upravit* na *Zobrazit* pro obyčejné *Hosty*, kteří nemají možnost provádět úpravy.

8.4.8 login.ctp

Úprava hlavičky. Stavové zprávy o přihlášení byly sjednoceny se stavovými zprávami celé aplikace a byly přemístěny pod horní menu. Přibylo zde tlačítko k registraci. Informace o prvním spuštění byly odstraněny.

8.4.9 register.ctp

Identický formulář jako přihlášení sloužící pro uživatelské registrace. Po registraci je zde automatické přesměrování na přihlášení, ale lze se tam dostat i samostatným tlačítkem.

8.4.10 profile.ctp

Nová šablona reprezentující správu profilu. Nabízí dynamický formulář pro změnu dostupných vlastností profilu a v případě *Administrátora* i editor dalších uživatelů. Ty je možné nově vytvořit, ve spodní tabulce přehledně vypsat či smazat nebo ve sdíleném formuláři i upravit.

8.5 Testování

8.5.1 Proces registrace

Byl otestován kompletní proces uživatelské registrace. Po prvním přihlášení je uživatel pochopitelně odpojen a automaticky přesměrován na nastavení profilu, kde si musí nakonfigurovat připojení.

Po každém uložení profilu je vynucený *restart* aplikace, kdy je uživatel přesměrován na základní stránku a je proveden pokus o nové připojení k databázi.

Při úspěšném připojení k databázovému serveru (platná adresa serveru, přihlašovací jméno i heslo) je proveden pokus o připojení k zvolené databázi. Pokud tato databáze neexistuje nebo neobsahuje platné tabulky prostředí EA, systém toto oznámí, ale již je k serveru připojen.

Pokud je správně vyplněna i databáze, je ještě nutné zaškrtnout volbu *[všechny]* v nastavení kořenů, jinak systém není schopen zobrazit stromovou strukturu v levé nabídce. I na toto systém upozorní.

Seznam dostupných kořenů se ovšem zobrazí až po úspěšném připojení k databázi. Poté je možné vybrat jeden či více kořenů, které se v levé stromové struktuře zobrazí jako výchozí.

8.5.2 Časová odezva vzdáleného připojení

V testech bylo ozkoušeno i připojení mimo mateřský *localhost*, který běžel pro vývojářské účely na běžném domácím počítači. Připojení bylo navázáno s centrálním databázovým serverem nasazeným na internetové páteři. Komunikace v takovémto případě dosahovala dlouhých prodlev cca 5 sekund na požadavek z důvodu pomalé rychlosti uploadu domácího počítače. Takovéto řešení tedy není doporučeno, protože celá aplikace je ve skutečnosti tlustým SQL klientem. Nicméně kromě rychlosti vše funguje naprosto bez problémů.

Zatímco připojení mezi dvěma vzdálenými, avšak na internetové páteři připojenými, servery probíhalo mnohem rychleji. Zde je odezva na požadavek pouhé dvě sekundy, což sice může vypadat velice dobře, avšak při běžné práci je to stále vcelku rušivý element. Bohužel tato prodleva asi nepůjde už žádným způsobem snížit, protože na *localhostu* je odpověď prakticky instantní.

8.6 Nasazení

Nasazení aplikace bylo v této iteraci minimalizováno a to především oproštěním nutnosti instalovat databázi prostředí EA, která je nyní zcela nezávislá. Provozovateli aplikace tak pouze stačí mít správnou verzi PHP a MySQL, nakopírovat systémové soubory na server, změnit konfigurační soubor *Config/database.php*, vyplnit do něj správné přihlašovací údaje k nově vytvořené prázdné databázi a aplikace již sama při prvním spuštění vytvoří potřebné tabulky a je ihned připravena k plnému provozu.

8.7 Zhodnocení

Výsledkem šesté iterace je výrazné rozšíření použitelnosti celé aplikace a přetransformování na službu. Po správném nasazení je pak vše již samoobslužné. *Administrátoři* se mohou samovolně registrovat a bez nějakého potvrzování si mohou sami nastavit příslušnou konfiguraci k jejich serverům, které mohou okamžitě spravovat. Dokonce mohou vytvářet i podřízené uživatele s odlišnými oprávněními nebo dokonce úplně odlišnými konfiguracemi. Toto považuji za zlomový bod vývoje.

Kapitola 9

7. iterace

9.1 Zadání iterace

Sedmá iterace by se měla soustředit především na drobná vylepšení co se použitelnosti týče. Vylepšení ergonomie a doladění detailů na již existující funkcionalitě. Toto vše reflektuje přání zadavatele, aby aplikace byla raději s méně funkcemi, ale perfektně odladěná a domyšlená. Jako hlavní bod této iterace bylo zadáno odstranění limitu pouze 5. úrovňové hloubky v levé stromové struktuře. Ta by měla nyní být i lépe ovladatelná.

9.2 Analýza

9.2.1 Současné problémy stromové struktury

V aktuální podobě se levá stromová struktura balíčků prostředí EA chová tak, že pomocí frameworku je načtena kompletně celá pomocí několika doslova obřích SQL dotazů, které jsou velice nepraktické. Možnosti automatického spojování modelů jsou zde u konce, protože se nacházíme v struktuře obsahující cykly a takřka neomezenou hloubku. Ta byla dosud stanovena explicitně na 5, protože větší hloubka již nebyla v časově pohodlných intervalech dosažitelná.

Omezení tak byly dvě. Jedna byla celá struktura vždy kompletně rozbalená (což může ale i nemusí být přehledné) bez možnosti se v ní nějak inteligentně pohybovat a dále nebylo možné se zanořit do větví hlubších úrovně 5. Naším úkolem je nyní oba tyto nedostatky odstranit.

9.2.2 Možná řešení

Při podobných problémech každého zcela automaticky napadá řešení v podobě postupného načítání jednotlivých větví. Jedná se v podstatě o praxí ozkoušené řešení, které se již široce používá. Jednotlivé větve budou zabaleny a zatím bez obsahu. Ten se začne načítat až po jejich otevření. Chvilku to bude trvat, protože to bude vyžadovat požadavek k serveru

na pozadí, ale výsledný efekt bude přesně dle očekávání. Navíc od této chvíle budou již nová data nacachovaná a zavření této větve a její znovuotevření již bude okamžité.

Problém je ale zcela jinde. Nacházíme se ve webovém prostředí, které funguje modelem *požadavek -> odpověď*, což znamená, že každý odkaz a většina kliknutí na různé objekty jsou ve skutečnosti znovunačtením celé webové stránky. Uživatel by se tak velice pohodlně orientoval ve stromové struktuře, avšak jakmile by nějaký balíček skutečně otevřel (myšleno nechal si načíst jeho obsah v pravém výpisu požadavků), stránka by se celá načetla znovu a celý strom by byl nejenom, že zase zavřený, ale bylo by ho nutné celý postupně znovu načítat!

Dlouho bylo zvažováno řešení v podobě rámců, zastaralé webové technologie, která by umožnila obnovit jen určitou část webové stránky, ale přišlo mi škoda takto celý framework ničit a zřejmě by s tím bylo i mnoho práce a muselo by se předělat spousta již hotových věcí.

Podobné, avšak modernější řešení, by pak bylo v podobě DHTML [14], někdy také alternativně nazývaného jako AJAX [11]. Princip i výsledek by však byl naprosto identický jako při použití rámců. Diskuze na internetu navíc poukazují, že dynamické načítání obsahu za pomocí rámců je dokonce rychlejší. ROzdílná by tak byla pouze implementace.

9.3 Návrh

9.3.1 Zvolené řešení

Nakonec bylo přece jenom vybráno řešení v podobě postupného načítání větví s výrazným vylepšením, které odstranilo všechny jeho nedostatky. První důležitou věcí je fakt, že přepracovaná stromová struktura bude plně kompatibilní s tou stávající, což umožní nejenom postupné načítání větví, ale i zobrazení již přednačtené struktury z dostupných dat.

Druhá invence pak přinesla využití nacachovaných dat a jejich propagaci do informací uložených v uživatelském sezení (tzv. PHP SESSIONS [5]). Jedna alternativa pracovala s návrhem, že by toto sestavování dat zajišťoval klientský JavaScript, ale zde by byl problém při přesunu dat zpátky na server. Proto se vše děje přímo v controlleru, který zpracovává a načítá jednotlivé úrovně větví pro dynamické zobrazování struktur stromu. Každou novou informaci systém automaticky „přilepí“ ke stávajícímu virtuálnímu stromu, který si udržuje v uživatelském sezení. Pokud pak uživatel obnoví stránku, tyto informace jsou stále dostupny a již načtený a otevřený strom se vykreslí bez prodlevy a lze se dále zanořovat tam, kde jsme skončili.

Jediná nepřesnost zde vzniká v případě, že uživatel některé již načtené větve stromu zase zavře. Tuto skutečnost si systém nikde neukládá (bylo by nutné zaslat extra požadavek směrem k serveru) a proto při obnově stránky jsou všechny již načtené větve plně rozbalené. Druhým problémem je skutečnost, že jak postupně všechny větve otevíráme, pracujeme pak už jen s nacachovanými a nikoliv skutečnými daty. Toto je v plánu vyřešit speciálním tlačítkem pro vyčištění celé stromové cache.

9.4 Implementace

9.4.1 AppController

V bázovém controlleru došlo jen k drobným úpravám v podobě přesunutí umístění načítání seznamu všech balíčků a přidáním speciální proměnné v SESSION s názvem *trees*, která je před každým průběhem prohlédnuta a obsahuje-li nějaká data, jsou použity namísto výchozích uživatelských kořenů pro vykreslení stromové struktury. Výchozí načítání stromové struktury bylo omezeno na hloubku úrovně 0.

9.4.2 EntityController

Oprava návratu do správného balíčku po úspěšném uložení upravované entity.

9.4.3 PackageController

Zcela nový controller určený k zpracovávání požadavků na pozadí pro dynamické rozšiřování stromové struktury. Nemá žádná vlastní view a obsahuje pouze čtyři metody, z toho dvě neveřejné.

tree(package , deep)

Veřejná metoda (akce) obsluhující samotné AJAX požadavky. Přebírá parametry udávající otcovský balíček a absolutní hloubku od výchozího kořene. Na základě těchto informací načte jednu úroveň vnořených podbalíčků a elementů, aktualizuje cache v sezení a vykreslí malý kousek stromové struktury.

clear()

Veřejná metoda (akce), která vymaže nacachovaná data v uživatelském sezení a přesměruje stránku na základní výpis všech požadavků při kompletně zabaleném stromě.

update(packages , entities , package)

Privátní funkce sloužící k aktualizaci cache. Nejprve přeformátuje entity do správného formátu, poté načte cache z uživatelského sezení a pro všechny výchozí kořeny provede rekurzivní aktualizaci. Výsledek opět uloží do uživatelského sezení. Funkce nemá návratovou hodnotu.

cache(cache , packages , entities , pid)

Privátní funkce, která projde všechny větve obsažené v aktuální úrovni, dokud nenalezne větev s *Package_ID = pid*. Takové větvi přiřadí nové potomky (*packages*) a entity (*entities*). Pokud požadovaná větev není nalezena, funkce pokračuje rekurzí sama na sebe o úrovni hlouběji pro každou aktuální větev, dokud není požadovaná větev nalezena nebo není dosaženo listů stromu. Jako parametr *cache* je očekávána reference, funkce nemá návratovou hodnotu.

9.4.4 RequirementController

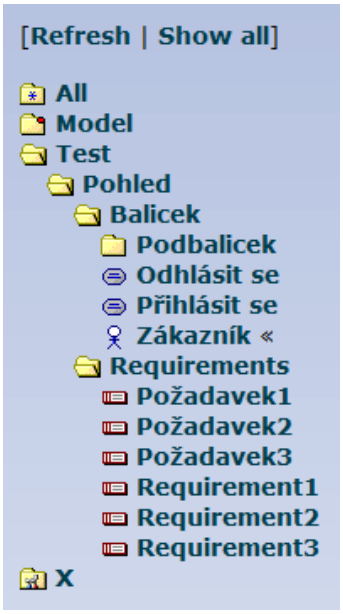
Oprava návratu do správného balíčku po úspěšném uložení upravovaného požadavku.

9.4.5 UsersController

Při změně konfigurace profilu je vynuceno vymazání stromové cache.

9.4.6 Výsledek stromové struktury a význam ikonek

Výsledné chování stromové struktury je velice uspokojivé a funguje přesně tak, jak bylo zamýšleno. Chování se nyní trochu změnilo. Pokud chceme manipulovat se samotným stromem, klikáme na ikonky jednotlivých balíčků. Pokud klikneme na jejich popisek, balíček se nám otevře v pravém výpisu požadavků. Následuje ilustrační obrázek se všemi použitými ikonkami balíčků, jejichž význam si nyní vysvětlíme.



Obrázek 9.1: Ukázka dynamické stromové struktury

Nahoře si můžeme všimnout stromové nabídky. Tlačítko *Obnovit* vymaže nacachovanou stromovou strukturu a vše zabalí. Tlačítko *Zobrazit vše* obnoví stránku a zobrazí úplný výpis všech požadavků v pravé části.

První balíček s názvem *All* s hvězdičkou je zavřený. Ona hvězdička symbolizuje, že ještě nebyl prozkoumán, takže systém neví, zda v něm něco je či je prázdný. Tuto skutečnost zjistíme kliknutím na něj.

Poslední balíček s názvem *X* se nachází ve stavu bezprostředně po kliknutí na balíček s hvězdičkou. Ikonka se změní na pracovní nástroje, což symbolizuje, že systém vyslal po-

žadavek na pozadí a načítá obsah balíčku. Tento stav trvá jen velice krátkou dobu v řádu milisekund a skončí sám příchodem obsahu balíčku.

Druhý balíček s názvem *Model* je takový balíček, o kterém systém zjistil, že je uvnitř prázdný. Tato skutečnost je dále uchovávána a balíček byl zbaven své události po kliknutí, protože k ní již není žádný další důvod. Toto je finální pozice, ze které se balíček již nemůže dostat.

Třetí balíček s názvem *Test* a několik dalších balíčků pod ním je otevřený a je zobrazena i jeho podstruktura. Stane se tak automaticky bezprostředně po načtení jeho dat nebo po jeho znovuotevření. Takovýto balíček můžeme kliknutím zase zavřít, čímž schováme celou jeho podstrukturu.

Šestý balíček s názvem *Podbalicek* je naproti tomu zavřený, ale již neobsahuje hvězdičku, takže o něm systém ví, že není prázdný. Do takového stavu se balíček dostane jedině zavřením. Může však být kdykoliv zase otevřen kliknutím na něj, čímž opět rozbalí svoji podstrukturu.

9.4.7 tree.ctp

V tomto view došlo k nejrozsáhlejším změnám. Většina klíčové funkce *RenderTree(obj , icons , root , deep)* byla předělána tak, aby byla schopna zpracovat jak statická, tak dynamická data, navíc musí umět správně rozpoznat druhy ikonek, které má jednotlivým balíčkům přiřadit a zda jsou právě otevřeny či ne. Dalších změn jsme se pak dočkali hlavně v JavaScriptu, kde přibyly čtyři následující funkce.

request(url , callback , data)

Převzatá funkce z mého předešlého projektu starající se o vytvoření objektu XMLHttpRequest a voláním AJAXového požadavku na pozadí. Funkce přebírá URL stránky zpracující požadavek, návratovou funkci a parametrická data.

response(request , callback)

Převzatá funkce z mého předešlého projektu starající se o zpracování návratového obsahu objektu XMLHttpRequest. Funkce přebírá tento objekt a návratovou funkci, které po zpracování předá výsledek.

tree(obj)

Klíčová funkce přebírající objekt HTML elementu, na který bylo právě kliknuto. Jedná se o prvky obsahující aktuální ikonku balíčku. Tato funkce nejprve načte všechny metadata z HTML5 atributů s prefixem *data-* samotného objektu a pak vlastní podvětev na základě rozhodnutí buď rozbalí, zabalí a nebo zašle AJAXový požadavek na její načtení, pokud tak dosud nebylo učiněno.

show(data , obj)

Návratová funkce pro AJAXové volání. Tato funkce zpracuje příchozí data a doplní je do předpřipraveného elementu. Na základě obsahu upraví balíčku jeho ikonku a je-li nějaký obsah dostupný, nechá balíček rozbalit.

9.4.8 index.ctp

Dílčí drobné úpravy a změna zobrazovaných sloupců ve výpisu požadavků.

9.4.9 edit.ctp a další šablony

Především kosmetické úpravy a omezení příliš dlouhých názvů balíčků, entit a požadavků pomocí funkce *TrimText(text , limit , more , tags)*, která byla převzata z mého předešlého projektu a umístěna globálně v souboru *Config/bootstrap.php*, což zajišťuje její globální dostupnost ve všech modelech, view i controllerech.

9.5 Testování

9.5.1 Chování dynamické stromové struktury

Nová dynamická stromová struktura byla pochopitelně několikrát otestována. Přes mnoho chyb a neduhů, které byly odstraněny se zde vyskytují pouze dva aktuální problémy, které nabízejí otázky, zda a jak je řešit. Jedna chyba je programátorská a sice občas se v dynamickém načítání objeví neznámé „fantom entity“, které se nepromítnout do nacachovaného stromu po obnovení stránky.

Druhý nedostatek je pak pouze logický. Stromová struktura je po obnovení stránky a načtení dat z cache kompletně celá rozbalena nezávisle na tom, zda uživatel měl nějaké již načtené podstruktury zavřené. Tato skutečnost má své řešení v ukládání stavu otevřenosti jednotlivých balíčků do uživatelského sezení pomocí speciálních AJAXových požadavků.

9.6 Zhodnocení

Výsledkem sedmé iterace je oprava mnoha drobných chyb, dopilování funkcionality a přiblížení celé aplikace představám zadavatele. Implementací dynamické stromové struktury se aplikace opět posunula dále k její finální podobě, která bude opět záviset na následující konzultaci se zadavatelem.

Kapitola 10

8. iterace

10.1 Zadání iterace

Osmá iterace by měla celý projekt úspěšně uzavřít. Je zde prostor pro opravy chyb nalezených při běžném používání, přidání posledních vylepšení a v plánu je i přidání podpory lokalizace včetně úplného českého překladu. Dále se tato poslední iterace bude zaměřovat hlavně na důkladné testování a poskytne i finální návod pro úspěšné nasazení.

10.2 Analýza

10.2.1 Zpětné reakce

Konzultace s klientem (zadavatelem projektu) v posledních týdnech vývoje bohužel nebyla natolik častá, jak by si projekt zasluhoval. Důvodem byla časová vytíženost obou strana především neočekávaná nemoc klienta. Proto zůstala funkcionalita, implementovaná v předchozích iteracích, neokomentovaná a bez konstruktivních připomínek, které by vedly k vypilování detailů k dokonalosti. Tato práce tak zůstala na realizátorovi.

10.2.2 Multijazyčnost

Projekt byl od začátku vyvíjen ve frameworku, který v sobě podporuje multijazyčnost s vizí vyhotovení ve dvou základních jazycích. V Češtině a technické Angličtině. Angličtina byla zvolena jako vývojový a Čeština potom jako výchozí jazyk. To znamená, že až do této chvíle je vše pouze v angličtině. Navíc v technické, napevno implementované napříč celým projektem. Tento jazyk je tedy jako jediný dostupný i bez jakýchkoliv dalších lokalizačních souborů. Tato angličtina pak může být v budoucnu nahrazena skutečnou profesionální angličtinou prostřednictvím lokalizace. Jako jediný přídavný jazyk tedy bude dodávána Čeština.

10.3 Návrh

10.3.1 Lokalizace

Jako výchozí jazyk celého systému je zvolena Čeština. Celý překlad je umístěn v souboru:

`Locale/ces/LC_MESSAGES/default.po`

Podobné lokalizační struktury jsou vytvořeny ještě pro skutečnou Angličtinu a Němčinu, ale jsou prázdné. Žádná lokalizace v nich není. Může být ale v budoucnu kdykoliv doplněna zkopírováním českého souboru *default.po*, který v *plaintextové* podobě obsahuje celý překlad. Zde se střídají po řádcích dvojice klíč a hodnota. Klíčem jsou *na znak přesné* podřetězce originální technické angličtiny a hodnotou pak samotný ekvivalent v daném jazyce. Následuje ukázka:

```
msgid "Hello"
msgstr "Ahoj"

msgid "Welcome"
msgstr "Vítejte"
```

Tento soubor může být kdykoliv editován v běžném *plaintextovém* editoru, avšak doporučuje se použít speciálních překladatelských nástrojů, např PoEdit[1]. Samotná souborová struktura a nacionální kódy pak podléhají mezinárodnímu standardu *ISO 639-2*. Dodržením tohoto standardu lze do projektu přidávat libovolné další lokalizace, které je ovšem nutné definovat v aplikačním controlleru a připravit pro ně příslušnou vlaječku.

10.3.2 Překlad

Celý překlad byl psán ručně včetně extrakce všech použitých klíčů (pouze výčtové seznamy prostředí EA byly extrahovány scriptem) a to právě z důvodu 100

Systém EAP ale občas využívá i dynamickou lokalizaci z proměnných, což mu umožňuje přeložit i veškeré nabídky prostředí EA. Jako příklad poslouží třeba typy jednotlivých objektů, které jsou uloženy v databázi EA a systémem EAP načítány jako možnosti k použití. Uživatel je ale vidí krásně česky přeložené, takže laik má lepší představu o tom, co znamenají. Zkušený UML modelář pak zajisté dá přednost původním anglickým výrazům. Nutno dodat, že v plnohodnotném prostředí EA se česky zvolené hodnoty nijak neprojeví. Ukládány jsou původní anglické výrazy.

10.4 Implementace

10.4.1 Přepínání jazyků

Systém jako takový pracuje standardně s technickým jazykem, což jsou textové řetězce předané funkci `__()` (dvě podtržítka). Tyto se zobrazují v případě, že není zvolena žádná lokalizace nebo zvolená lokalizace neexistuje a nebo pouze pro daný řetězec chybí ekvivalentní překlad. Toho je využito při přepnutí na Angličtinu, kde se (stejně jako v případě Němčiny) namísto překladu (který chybí), zobrazí technická lokalizace, což v případě Angličtiny vypadá jako skutečný překlad.

[1] <http://poedit.net/>

Vše podstatné se děje v, dříve již probírané, funkci *beforeFilter()* v hlavním *AppControlleru*. Třída samotná obsahuje veřejné proměnné *langs* s výčtem možných lokalizací a *lang* s výchozí lokalizací.

Následuje ukázka lokalizačního kódu:

```
$this->set('langs' , $this->langs);
if (!Configure::read('Config.language')) {
    Configure::write('Config.language' , $this->lang);
}
if (IsSet($_GET['lang'])) {
    $this->Session->write('Config.language' , $_GET['lang']);
}
if ($this->Session->check('Config.language')) {
    $this->lang = $this->Session->read('Config.language');
    Configure::write('Config.language' , $this->lang);
}
$this->set('lang' , $this->lang);
```

Nejprve je propagován seznam dostupných lokalizací do view. Dále v případě, že systém nemá nastavenou žádnou lokalizaci, je zvolena ta výchozí. Je-li nastaven GET parametr *lang*, uloží se do právě vytvořeného uživatelského sezení jeho hodnota pod stejným názvem. Je-li dostupná nějaká lokalizace v uživatelském sezení, zvolí se jako výchozí a systém se do ní přepne. Nakonec se výsledná lokalizace opět propaguje do view.

Implementace ve view pak je již velice jednoduchá. Vytvoří a ostyluje se blok s nabídkou lokalizací a iteruje se přes všechny dostupné lokalizace. To způsobí vykreslení obrázků s národními vlajkami. Vybraný jazyk je barevně odlišen. Po kliknutí na vlajky je vytvořen GET požadavek s parametrem *lang* o příslušné hodnotě.

10.5 Testování

10.5.1 Testování vůči zadání

Tento test má za úkol zhodnotit výsledný produkt a jeho atributy vzhledem k jeho počáteční specifikaci. Jako specifikaci použijeme zadání.

Test	Zkontrolování programu vůči zadání.
Zadání	Výsledek
Vytvořit webovou aplikaci.	Splněno.
Prohlížení požadavků.	Splněno.
Tvorba požadavků.	Splněno.
Úprava požadavků.	Splněno.
Správa souvisejících elementů.	Jejich zobrazení.
Grafické zvýrazňování změn požadavků.	Toto volitelné zadání nakonec klientem nebylo požadováno.
Zobrazování požadavků přímo v diagramu.	Splněno.
Editace požadavků přímo v diagramu.	Toto volitelné zadání nakonec klientem nebylo požadováno.
Generování dokumentace.	Toto volitelné zadání nakonec klientem nebylo požadováno.
Zadání řešit projektovým způsobem.	Splněno.
Využít iterativní vývoj.	Splněno.
Publikovat pod BSD licencí.	Splněno.
Využít jazyk UML.	Splněno.

Test	Zkontrolování programu vůči zadání.
Použít nástroj Enterprise Architect.	Splněno.
Nahrání dokumentace na Assemblu.	Splněno.
Vykazovat odpracované hodiny.	Doplněno.
Jednoduchost.	Splněno.
Intuitivnost ovládání.	Splněno.

10.5.2 Testování vůči případu užití

Tento test má za úkol otestovat správnost programu vůči jeho specifikaci a požadavkům uživatele. Provedeme tedy krok po kroku jeden ze scénářů případů užití.

Test	Provedení scénáře č. 2 v sekci Scénáře případů užití 2.6.
Výsledek	Scénář byl uplatněn na oba typy požadavků. Vše proběhlo přesně podle předpokladu. Systém úspěšně vrací uživatele do dříve otevřeného balíčku. Cache stromové struktury navíc také funguje v pořádku.

10.5.3 Test odezvy

Tento test má za úkol otestovat čas potřebný k vykonání jednotlivých požadavků za různých podmínek.

Test	Načtení úvodní stránky.
Výsledek	250 - 350ms
Test	Načtení lokální databáze malého projektu.
Výsledek	300 - 400ms

Test	Načtení vzdálené databáze se serverovou konektivitou malého projektu.
Výsledek	400ms
Test	Načtení vzdálené databáze s běžnou konektivitou malého projektu (pomalý upload).
Výsledek	800ms.
Test	Načtení vzdálené databáze s běžnou konektivitou malého projektu (pomalý download).
Výsledek	1000ms.

10.5.4 Zátěžový test

Tento test má za úkol otestovat program a jeho odezvu při nedostatku systémových zdrojů.

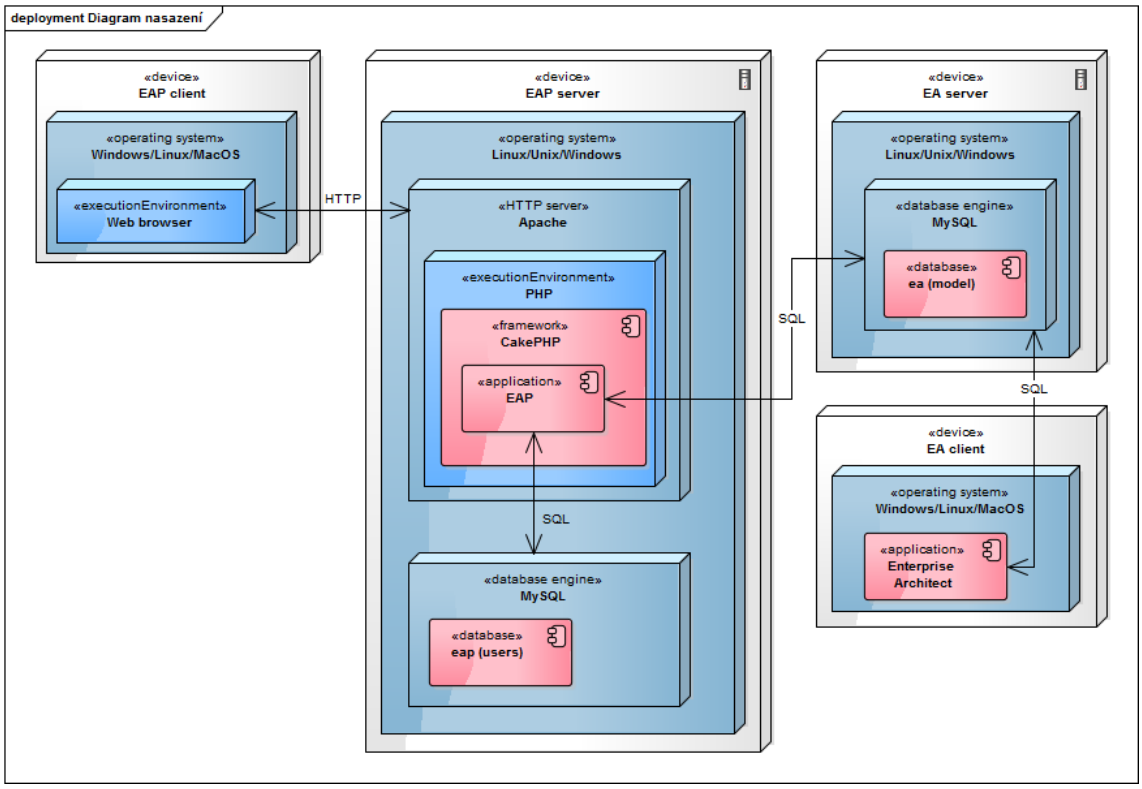
Test	Spuštění náročného SQL příkazu na pozadí.
Výsledek	Odezva 3500ms.

10.6 Nasazení

10.6.1 Postup při nasazení finální aplikace do reálného provozu

Pro lepší pochopení komunikačních struktur hned na úvod přikládám názorný diagram nasazení, demonstrující obecné prostředí čtyř samostatných zařízení:
Diagram se skládá ze čtyř následujících fyzických zařízení:

1. **EAP server** - Hlavní server, na kterém běží samotná aplikace EAP a její databáze uživatelských nastavení. Celá kapitola nasazení se bude týkat výhradně tohoto zařízení. Aplikace EAP komunikuje prostřednictvím SQL jak s databázovým poskytovatelem EA serveru, tak s lokálním databázovým poskytovatelem.
2. **EAP client** - Libovolný klientský počítač s libovolným operačním systémem. Podmínkou je pouze aktuální verze libovolného webového prohlížeče. Tento prohlížeč komunikuje skrze protokol HTTP s webovým serverem Apache, běžícím na EAP serveru.



Obrázek 10.1: Diagram nasazení

3. **EA server** - Libovolný počítač (klient či server) s databázovým serverem MySQL s platnou instancí EA databáze. EA server by měl být s EAP serverem ideálně spojen alespoň 100Mbitovým symetrickým připojením.

4. **EA client** - Libovolný klientský počítač s libovolným operačním systémem. Podmínkou je pouze aktuální verze prostředí EA. EA client by měl být s EA serverem ideálně spojen alespoň 100Mbitovým symetrickým připojením. Většinou to ale bývají identické počítače. Prostředí EA komunikuje s databázovým poskytovatelem EA serveru prostřednictvím SQL.

Toto uspořádání představuje pouze modelovou situaci. Většinou bývá uzel EA serveru identický s uzlem EA clienta. V některých případech pak může na stejném uzlu běžet i EAP server a teoreticky i EAP client, ale toto řešení už rozbíjí celý zamýšlený koncept nasazení.

Následující tabulka prezentuje minimální výchozí požadavky pro provozování aplikace EAP:

Hardware	*Nejsou kladeny žádné speciální nároky na hardware.*
Konektivita	Alespoň symetrický 100Mbit.
Operační systém	Linux, Unix, Windows.
Webový server	Apache (Windows: 2.2.25, Linux: 2.2.17)
Databázový engine	MySQL (Windows: 5.6.20, Linux: 5.1.61)
Běhové prostředí	PHP verze 5.2.8 nebo novější.
Framework	CakePHP verze 2.5.5.

A nakonec samotný postup, jak aplikaci EAP nasadit:

1. Obsah instalačního balíčku rozbalíme někam do veřejných složek webového serveru (nejčastěji */var/www/*) buď libovolně nebo do podsložky *app* frameworku CakePHP.
2. Otevřeme soubor *webroot/index.php* a nalezneme řádek s definicí cesty jádra frameworku. Bude vypadat takto: *Define('CAKE_CORE_INCLUDE_PATH', ROOT.DS.'cakephp'.DS.'lib');*. Cestu upravíme, aby směřovala na složku *lib* frameworku CakePHP. (Výchozí nastavení je: *./../../cakephp/lib*.)
3. Otevřeme soubor *Config/database.php* a upravíme hodnoty *login*, *password* a *database* příslušnými hodnotami k připojení k lokálnímu SQL serveru. Databázi si vytvoříme separátně s kódováním *utf8_general_ci*.
4. Spustíme aplikaci navštívením její adresy prostřednictvím protokolu HTTP.

10.7 Zhodnocení

Výsledkem osmé iterace je hotová aplikace EAP. Byla přidána lokalizace, řazení požadavků a bylo opraveno spoustu drobných chyb. Aplikace podstoupila několik testů různých druhů a v tuto chvíli se jeví jako stabilní. Také byl poskytnut detailní návod pro její reálné nasazení.

Kapitola 11

Závěr

11.1 Ukončení vývoje

Celý vývojový proces byl nakonec přesně po 12 týdnech ukončen. Za tuto dobu se stihlo realizovat 8 z celkem 10 plánovaných iterací. Tento rozdíl však vůbec neznamená, že by se něco nestihlo nebo nedokončilo. Vývoj byl od samého začátku koncipován jako iterativní s dynamickým směrem, který záležel na konzultacích s klientem. Velká část posledních iterací byla navíc věnována volitelným částem zadání, pro jejichž realizaci se klient nakonec nerozhodl. Místo toho se zapracovalo na věcech mnohem důležitějších a osobně si myslím, že tento přístup aplikaci velice prospěl. Iterativní vývoj kombinovaný s průběžnými konzultacemi byl tedy velice dobrá volba.

11.2 Shrnutí výsledné aplikace

Výsledkem celého vývoje je hotová aplikace EAP ve verzi 1.0. To zde zmiňuji záměrně, protože už na začátku vývoje se předpokládalo, že se bude aplikace někdy v budoucnu dále rozvíjet.

Byla vytvořena webová aplikace, umožňující prohlížení, tvorbu i úpravu požadavků (včetně souvisejících elementů) evidovaných v databázi nástroje Enterprise Architect. Jako vedlejší funkce (implementovaná trochu nejasným zadáním) je zde navíc možnost manipulovat s požadavky dvou různých druhů, což je určitě přidaná hodnota, která se neztratí.

O grafické zvýrazňování změn požadavků zadání rozšířeno nebylo, aplikace jej tedy neobsahuje. Zobrazení požadavků v diagramu však implementováno bylo. O generování dokumentace zadání rozšířeno také nebylo.

Celkově však byla aplikaci věnována dostatečná péče na to, aby každá její funkcionalita byla spolehlivá a dotažená nejenom do konce, ale aby se i dobře a snadně používala. Bylo také přidáno mnoho dalších funkcí, především samostatná registrace uživatelů a konfigurace vzdálených připojení k libovolným instancím databáze EA. Toto jsou funkce v původním zadání vůbec nezmíněné, avšak jejich efekt na výsledek je naprosto klíčový. Díky nim se dá

celá aplikace využít a provozovat jako služba pro neomezený počet uživatelů, kteří si každý spravují své oddělené projekty.

Celý projekt byl řešen projektovým způsobem s využitím iterativního vývoje pod BSD licencí. Při tvorbě dokumentace byly využity standardy UML a nástroj EA, ze kterého pochází všechny publikované diagramy. Dokumentace je volně dostupná na FREE PUBLIC ASSEMBLA PROJEKTu na adrese <https://www.assembla.com/spaces/eap-2014/wiki>. Aplikace je velmi jednoduchá, avšak plně funkční a pro uživatele velice intuitivní. Přesně tak, jak původní zadání požadovalo.

11.3 Naplnění stanovených cílů a konečné zhodnocení

Za velice pozitivní věc vidím fakt, že vše co bylo požadováno bylo nakonec i realizováno. Alespoň z pohledu klienta, který průběžně modifikoval zadání dle svých aktuálních představ založených na průběhu vývoje. Jsem tedy velice rád, že na konci nemusím omlouvat absenci určitých funkcí, popřípadě upozorňovat na nedodělky. Zároveň však ale přiznávám, že zlepšit se dá všechno a potenciál k dalšímu rozšíření je zde obrovský. Z mého pohledu tedy všechny stanovené cíle byly zodpovědně naplněny.

Osobně tedy aplikaci hodnotím velice kladně, dobře se mi na ní pracovalo. Během vývoje nenastaly žádné neřešitelné komplikace. Všechny problémy byly nakonec úspěšně překonány do výsledné podoby.

11.4 Přínos aplikace

Protože je aplikace vyvíjena úplně od začátku, tak po splnění jejich stanovených cílů je sama o sobě velikým přínosem. Za největší přidanou hodnotu však pokládám způsob jejího provozování jako veřejné služby všem, kterým by mohla být užitečná, navíc dle licence zcela zdarma. Kdokoliv jí může nasadit a na svém veřejném či soukromém serveru neomezeně provozovat. Ke každé této instanci se může připojovat neomezený počet uživatelů, kteří jí pak dále budou plně využívat, aniž by výraznějším způsobem zatížili samotný server, který je jakýmsi prostředníkem. Veškerá komunikace se totiž odehrává zejména mezi uživatelským prohlížečem a vzdálenou databází EA. Samotná konfigurace je navíc plně samoobslužná a tak aplikace nepotřebuje žádnou pravidelnou správu či údržbu. Všechny tyto funkce navíc v původním zadání vůbec nebyly zaneseny.

11.5 Výhledy do budoucna

Během celého vývoje byly postupně sbírány drobné nápady a podněty k dalšímu rozvoji celého projektu. Vyskytují se mezi nimi i slepé vývojové větve, které mohou být dále někam rozvinuty. Toto vše bylo zaneseno pomocí ticketů do projektového prostoru na Assemble. Následuje tedy jejich kompletní výčet:

1. **Grafické zvýrazňování změn požadavků od určitého termínu či verze (inspirace z Microsoft Word).**

   Původní volitelná součást zadání.

2. **Zobrazování a editace požadavků přímo v diagramu.**

   Toto by mohlo nasadit pomyslnou korunu v uživatelské přívětivosti.

3. **Grafické zlepšení diagramů.**

   Implementace diagramů je v aplikaci zatím na úplném začátku. Škoda, že nedostala při vývoji větší prostor.

4. **Generování dokumentace (RTF/LATEX/,...).**

   Původní volitelná součást zadání.

5. **Rozšíření manipulace s požadavky.**

   Aplikace by mohla obsahovat více druhů a možností při manipulacích se samotnými požadavky.

6. **Modernější vzhled a ikonky.**

   Nejen diagram, ale celá grafická stránka projektu má veliký potenciál k vylepšení.

7. **Otestovat chování aplikace na velkém projektu (vymýtit fantom entity).**

   Bylo by dobré znovu zanalyzovat chování aplikace na velkém reálném projektu. Vedlo by to k odstranění drobných chyb.

8. **Persistence stavů větví při cachování stromové struktury.**

   Další bod podporující uživatelskou přívětivost.

9. **Accounting požadavků.**

   Nápad přidání nové funkcionality do aplikace.

Literatura

[1] CAKE SOFTWARE FOUNDATION, I. *CakePHP* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://cakephp.org/>.

[2] FOUNDATION, T. A. S. *Apache HTTP Server Project* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://httpd.apache.org/>.

[3] GROUP, T. P. *PHP Data Objects* [online]. 2014. [cit. 30. 11. 2014]. Dostupné z: <http://php.net/manual/en/book.pdo.php>.

[4] GROUP, T. P. *PHP* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://php.net/>.

[5] GROUP, T. P. *Session Handling* ¶ [online]. 2014. [cit. 2. 12. 2014]. Dostupné z: <http://php.net/manual/en/book.session.php>.

[6] LTD., C. *Ubuntu* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://www.ubuntu.com/>.

[7] MICROSOFT. *Windows* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://windows.microsoft.com/cs-cz/windows/home>.

[8] OMG. *Unified Modeling Language* [online]. 2014. [cit. 25. 10. 2014]. Dostupné z: <http://www.uml.org/>.

[9] ORACLE. *MySQL* [online]. 2014. [cit. 2. 11. 2014]. Dostupné z: <http://www.mysql.com/>.

[10] ORACLE. *MySQL Forums* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://forums.mysql.com/>.

[11] Přispěvatelé Wikipedie. *AJAX* [online]. 2014. [cit. 2. 12. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/AJAX>.

[12] Přispěvatelé Wikipedie. *API* [online]. 2014. [cit. 11. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/API>.

[13] Přispěvatelé Wikipedie. *CRUD* [online]. 2014. [cit. 1. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/CRUD>.

[14] Přispěvatelé Wikipedie. *DHTML* [online]. 2014. [cit. 2. 12. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/DHTML>.

[15] Přispěvatelé Wikipedie. *Domain Name System* [online]. 2014. [cit. 30. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Domain_Name_System>.

[16] Přispěvatelé Wikipedie. *GIF* [online]. 2014. [cit. 11. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/GIF>.

[17] Přispěvatelé Wikipedie. *Globally unique identifier* [online]. 2014. [cit. 11. 11. 2014]. Dostupné z: <http://en.wikipedia.org/wiki/Globally_unique_identifier>.

[18] Přispěvatelé Wikipedie. *Hypertext Transfer Protocol* [online]. 2014. [cit. 2. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol>.

[19] Přispěvatelé Wikipedie. *IP adresa* [online]. 2014. [cit. 30. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/IP_adresa>.

[20] Přispěvatelé Wikipedie. *Microsoft Windows Installer* [online]. 2014. [cit. 10. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Microsoft_Windows_Installer>.

[21] Přispěvatelé Wikipedie. *Model-view-controller* [online]. 2014. [cit. 10. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Model-view-controller>.

[22] Přispěvatelé Wikipedie. *Open Database Connectivity* [online]. 2014. [cit. 10. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Open_Database_Connectivity>.

[23] Přispěvatelé Wikipedie. *Objektově relační mapování* [online]. 2014. [cit. 8. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Objektově_relační_mapování>.

[24] Přispěvatelé Wikipedie. *Portable Network Graphics* [online]. 2014. [cit. 11. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Portable_Network_Graphics>.

[25] Přispěvatelé Wikipedie. *Requirement* [online]. 2014. [cit. 8. 11. 2014]. Dostupné z: <http://en.wikipedia.org/wiki/Requirement>.

[26] Přispěvatelé Wikipedie. *SQL* [online]. 2014. [cit. 30. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/SQL>.

[27] Přispěvatelé Wikipedie. *Uniform Resource Locator* [online]. 2014. [cit. 30. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Uniform_Resource_Locator>.

[28] Přispěvatelé Wikipedie. *Work breakdown structure* [online]. 2014. [cit. 8. 11. 2014]. Dostupné z: <http://cs.wikipedia.org/wiki/Work_breakdown_structure>.

[29] (W3C), W. W. W. C. *CSS3* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://www.w3schools.com/css/css3_intro.asp>.

[30] (W3C), W. W. W. C. *HTML5* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://www.w3schools.com/html/html5_intro.asp>.

[31] (W3C), W. W. W. C. *JavaScript* [online]. 2014. [cit. 9. 11. 2014]. Dostupné z: <http://www.w3schools.com/js/>.

Příloha A

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CRUD Create, Read, Update, Delete

DHTML Dynamické HTML

DNS Domain Name System

EA Enterprise Architect

EAP Enterprise Architect Plugin

GIF Graphics Interchange Format

GUID Globally Unique IDentifier

HTTP HyperText Transfer Protocol

IP Internetový Protokol

MSI MicroSoft Installer

MVC Model View Controller

ODBC Open DataBase Connectivity

ORM Objektově Relační Mapování

PDO PHP Data Objects

PHP PHP: Hypertextový preprocesor

PNG Portable Network Graphics

SQL Structured Query Language

UML Unified Modeling Language

URL Uniform Resource Locator

WBS Work Breakdown Structure

Příloha B

Obsah přiloženého CD

Na přiloženém CD se nachází PDF dokument s identickým obsahem této bakalářské práce v tiskové verzi pojmenovaný jako **BAP_gresjan_2014_2.pdf**. Jediný rozdíl je v absenci šablony pro tvorbu vazebných desek.

Všechny zdrojové soubory použité k vygenerování tiskové verze této práce pro prostředí LaTeXjsou pak umístěny ve složce **zdroje**.

Systém samotný se pak nachází ve složce **eap**, kde naleznete následující souborovou strukturu:

- **[Config]** - složka s konfiguračními soubory prostředí CakePHP
- **[Console]** - složka obsahující konzolové scripty aplikace
- **[Controller]** - složka obsahující všechny controllery aplikace
- **[Install]** - složka obsahující instalační SQL soubory
- **[Lib]** - složka s externími PHP knihovnami
- **[Locale]** - složka obsahující lokalizace
- **[Model]** - složka obsahující všechny modely aplikace
- **[Plugin]** - složka s externími zásuvnými moduly
- **[Test]** - složka s testovacími scripty
- **[tmp]** - složka pro dočasné soubory prostředí CakePHP
- **[Vendor]** - složka s dalšími externími moduly
- **[View]** - složka obsahující všechny view aplikace
- **[webroot]** - kořenová složka aplikace pro webový server
- **.htaccess** - konfigurační soubor pro webový server Apache
- **index.php** - hlavní vstupní soubor aplikace