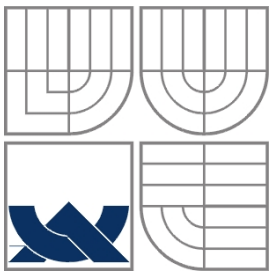
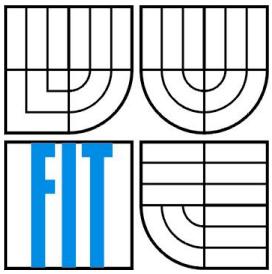


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÉ VÝVOJÁŘSKÉ NÁSTROJE PRO SYSTÉM CACHÉ

WEB DEVELOPMENT TOOLS FOR CACHÉ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN LOLA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. BURGET RADEK, Ph.D.

BRNO 2013

Abstrakt

Cílem této práce je vytvoření webového vývojářského nástroje pro systém Caché. Tento nástroj bude umožňovat vytváření a editaci tříd, rutin a metod. Při tvorbě bylo dbáno na to, aby výsledná aplikace využívala nastavení uživatelských práv ze systému Caché, a aby byla jednoduše šířitelná i rozšířitelná. Aplikace bude využívat skriptovací jazyk ObjectScript systému Caché a obvyklé webové technologie (HTML, CSS, Javascript). Text této práce popisuje kompletní vývojový cyklus aplikace od rozboru současného stavu, přes popis potřebných částí systému Caché, analýzu požadavků a návrh systému až po implementační detaily a rozbor testování a dosažených výsledků.

Abstract

The aim of this thesis is to create web developer tools for the Caché system. This tool allows the creation and editing of classes, routines and methods. Particular focus was given to the use of the user roles from the Caché system, the extensibility and easy installation. The application is implemented in the Caché ObjectScript language and uses common web technologies (HTML, CSS and Javascript). The text of this thesis describes the complete development cycle of the application from an analysis of the current situation through the description of the needed parts of Caché and the requirements analysis, system design to the implementation details and testing and analysis of results.

Klíčová slova

Webové vývojářské nástroje, vývojářské nástroje, Caché, systém Caché, databáze Caché, objektová databáze, postrelační databáze, editace tříd, editace metod, editace rutin, Intersystems, ObjectScript

Keywords

Web developer tools, developer tools, Caché, Caché systém, Caché database, object database, postrelational database, class editation, method editation, routine editation, Intersystems, ObjectScript

Citace

Martin Lola: Webové vývojářské nástroje pro systém Caché, bakalářská práce, Brno, FIT VUT v Brně, 2013

Webové vývojářské nástroje pro systém Caché

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.

Další informace o implementačních detailech mi poskytl zástupce společnosti Intersystems pan Ing. Daniel Kutáč.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Lola
2013

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Radkovi Burgetovi, Ph.D. za jeho odbornou pomoc a cenné informace a zkušenosti při tvorbě této práce.

Taktéž bych rád poděkoval panu Ing. Danielovi Kutáčovi za odborné rady týkající se implementace systému.

Obsah

1 Úvod

Objektový návrh je dnes již zažitým a nejvíce používaným modelem pro tvorbu komerčně úspěšných aplikací. Naproti tomu většina databázových systémů staví na relačním přístupu k datům. Softwaroví vývojáři tedy musí řešit dilema, jak objektově orientovaná data uloží a přečtou z relační databáze. Tento problém se snaží řešit databáze postrelačního typu a právě jedním z těchto typů databáze je i systém Caché.

Společnost Intersystems představila Caché v roce 1997 a tento systém se od té doby stal nejrozšířenější objektovou databází. Caché se využívá nejčastěji v nemocnicích pro ucelenou správu všech elektronických záznamů o pacientech. [2]

Vývoj probíhá dodnes a nyní se pod pojmem *systém Caché* rozumí objemný balíček aplikací, jehož součástí je postrelační databáze (povoluje objektový i relační přístup k datům), nástroje umožňující přímou manipulaci s daty, překladače pro soubor několika programovacích jazyků a konečně i webové aplikace pro správu tohoto databázového systému. [2] Tyto aplikace jsou zaměřeny převážně na správu uživatelů a jejich rolí, správu databází, jmenných prostorů a zabezpečení systému. Pro editaci a vytváření nových struktur, tříd, metod, rutin apod. je možné využít pouze desktopové aplikace (rok 2013).

Cílem této práce je vytvořit webový nástroj pro systém Caché, který umožní vytváření a editaci tříd, metod a rutin. Uživatel vytvořenou aplikaci nainstaluje na webový server se systémem Caché a následně získá přístup ke zdrojovým kódům i přes webové rozhraní.

Kapitola 2 popisuje systém Caché poněkud detailněji – dozvíme se v ní, jak systém funguje a jaké programovací jazyky využívá. V kapitole 3 se seznámime s již existujícími nástroji pro editaci zdrojového kódu v systému Caché. Jednotlivé aplikace (desktopové, webové) si krátce popíšeme. Kapitola 4 určí datové typy (dále jen *entity*), které vytvořená aplikace bude používat. Tyto entity představují vnitřní reprezentaci dat a jejich uspořádání v systému. S takto zmapovanou aktuální situací můžeme v kapitole 5 přejít k přesnější specifikaci zadání, jejíž součástí bude i use-case diagram a požadavky na zabezpečení vytvořené aplikace. S těmito znalostmi již můžeme navrhnout samotný webový nástroj pro editaci a vytváření entit i s uživatelským rozhraním (kapitola 6). V následující kapitole si projdeme implementační detaily a omezení, se kterými jsem se při vývoji potýkal. 8. kapitola se věnuje třem fázím testování aplikace, kde si jednotlivé fáze popíšeme a zhodnotíme jejich výsledky a přínos. Závěrečná kapitola 9 shrnuje dosažené výsledky, známe omezení aplikace a možnosti dalšího vývoje.

2 Postrelační databáze Intersystems Caché

Společnost Intersystems nazývá systém Caché *postrelační* databází. Tento pojem je ale příliš obecný a jeho význam si vyjasníme v této kapitole. Dále si obecně popíšeme části systému a programovací jazyky, které jsou v systému využívány.

Popis celého systému Caché by byl vzhledem k omezenému rozsahu této práce příliš obsáhlý, proto systém popíšu pouze obecně a detailněji zpracuji pouze ta témata a části systému, která úzce souvisí s tématem práce.

2.1 Popis systému

Pojem *postrelační* popisuje tu skutečnost, že systém vznikl až po databázích relačního typu. Je tedy třeba ujasnit si, že systém Caché využívá jak objektový, tak i čistě relační přístup k datům. Unifikovaná datová architektura poskytuje jednu společnou vrstvu pro oba přístupy. Preference je ale u objektového přístupu, výsledný zdrojový kód je potom přehlednější a čistší.

Caché v sobě ukrývá datový i aplikační server. Kvůli jednotnému přístupu k datům i k aplikační logice je právě aplikační logika uložená v podobě tabulek v databázi na serveru. [2] Tato informace a dále existence vestavěného API Caché dala možnost vzniku této bakalářské práce.

2.2 API systému Caché

Přístup do dokumentace API uživatel získá s instalací systému. Toto API je dostupné i online [3]. Zajímavostí je, že dokumentace API není vytvořena staticky, ale tvoří se dynamicky za chodu systému. Při přístupu do dokumentace se ze systému načtou žádané entity a uživateli jsou zobrazeny se všemi svými vlastnostmi, popisem a vzájemným propojením. Tento přístup hodnotím jako vysoce uživatelsky přívětivý – značně usnadňuje orientaci v cizím zdrojovém kódu bez toho, aby vývojář musel psát nějakou formu dokumentace navíc.

Znalost a využití API systému je pro tuto práci přímo stěžejní. Caché v sobě již obsahuje rozhraní pro vytváření a modifikaci svých entit – rutin, tříd a metod (bližší specifikaci a rozšíření seznamu entit se věnuji v kapitole 4). Systém samotný toto rozhraní využívá pro export, import a generování nových entit.

Detaily zpracování a implementace API jsou probrány v kapitole 8.

2.3 Programovací jazyky v Caché

Caché v sobě obsahuje několik programovacích jazyků, jako je např. Class Definition Language, Caché Basic, ObjectScript a další. Protože hlavním skriptovacím jazykem Caché je ObjectScript, je v něm napsána většina dostupných aplikací a tutorialů, a Intersystems na svých stránkách zaručuje jeho kompatibilitu skrz svými aplikacemi[1], budu aplikaci vytvářet právě v tomto jazyku.

ObjectScript je původním skriptovacím jazykem Caché. Je objektově orientovaný a nevyžaduje, aby byl zdrojový kód součástí metody. [1] Této možnosti využiji při tvorbě CSP stránky (bližší popis je v následující podkapitole).

2.4 Struktura aplikací

V této kapitole si popíšeme hlavní prvky využívané pro tvorbu aplikací v Caché. Opět platí, že prvků je více – popíšu pouze ty, které se týkají tématu práce.

2.4.1 Třídy

Třídy jsou v systému Caché základní jednotkou pro tvorbu rozsáhlých aplikací, jak je tomu ostatně i u jiných objektových systémů. Tyto třídy mohou obsahovat aplikační logiku, pracovat s databází i spoluvytvářet strukturu obsahu pro zobrazení uživateli. Systém Caché k těmto obvyklým funkčnostem přidává další speciální možnosti, jejichž zpracování ale není předmětem této práce a při vytváření aplikace je nevyužiji, proto jsem se rozhodl je zde neuvádět.

Při práci s třídami je třeba mít na paměti, do kterého jmenného prostoru spadají. V rámci tohoto jmenného prostoru má třída jedinečný název. Třídy lze dělit do tzv. balíčků. Tyto balíčky poté tvoří hierarchii v diagramu tříd.[2]

2.4.2 CSP stránka

Základní myšlenkou CSP stránky je rozdělení aplikace do více vrstev. Třídy by měly obstarávat správu dat, business logiku (např. kontrola, jestli je možné vykonat akci, validační pravidla pro akce, atd.) a provázání funkčnosti aplikace a CSP stránka se stará o zobrazení a reprezentaci dat dodaných právě ze tříd.[2]

3 Aplikace pro správu Caché

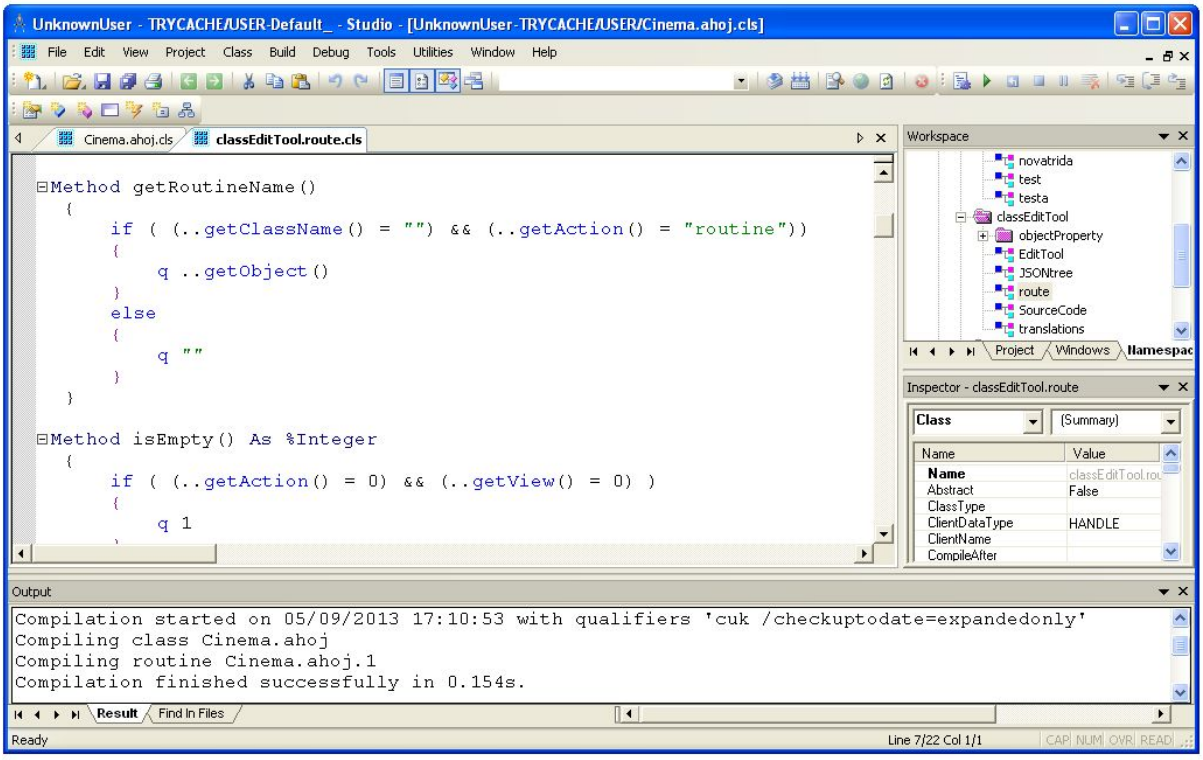
Než se vůbec pustíme do analýzy požadavků a návrhu systému samotného, bude vhodné seznámit se s již existujícími aplikacemi pro správu systému Caché. Nejdříve se zaměříme na desktopové aplikace, poté se pokusíme najít dostupné webové nástroje, nebo alespoň nalézt takové, které se svým zaměřením podobají účelu vytvářeného systému.

3.1 Desktopové aplikace

Součástí instalace systému Caché je vývojové prostředí Caché Studio a příkazová konzole Caché Terminal.

3.1.1 Caché Studio

Jediným bezplatným desktopovým nástrojem je Caché Studio.



Obrázek 1: Caché Studio

Tato aplikace je plnohodnotným vývojovým prostředím. Nabízí přehled a editaci všech entit, které lze v Caché vytvořit (rutiny, třídy a jejich vlastností, metody a další vztahované entity, csp stránky, apod.). Propracovanou částí je přidávání nových entit, které lze vkládat pomocí různých průvodců. V těchto uživatelsky přívětivích průvodcích lze navolit typ žádané entity a její další

vlastnosti. Druhou možností, jak přidat nové entity, je napsat tyto entity přímo do zdrojového kódu systému Caché.

Caché Studio na druhou stranu nabízí plno možností a přehledů, které ani znalý uživatel příliš často nevyužije. Proto bych vyzdvihl možnost skrytí prvků v editoru, tedy uživatelského nastavení rozložení prvků v aplikaci. Dále je třeba povšimnout si panelu s výsledky překladu, a přehledu tříd spolu se vztahovanými entitami.

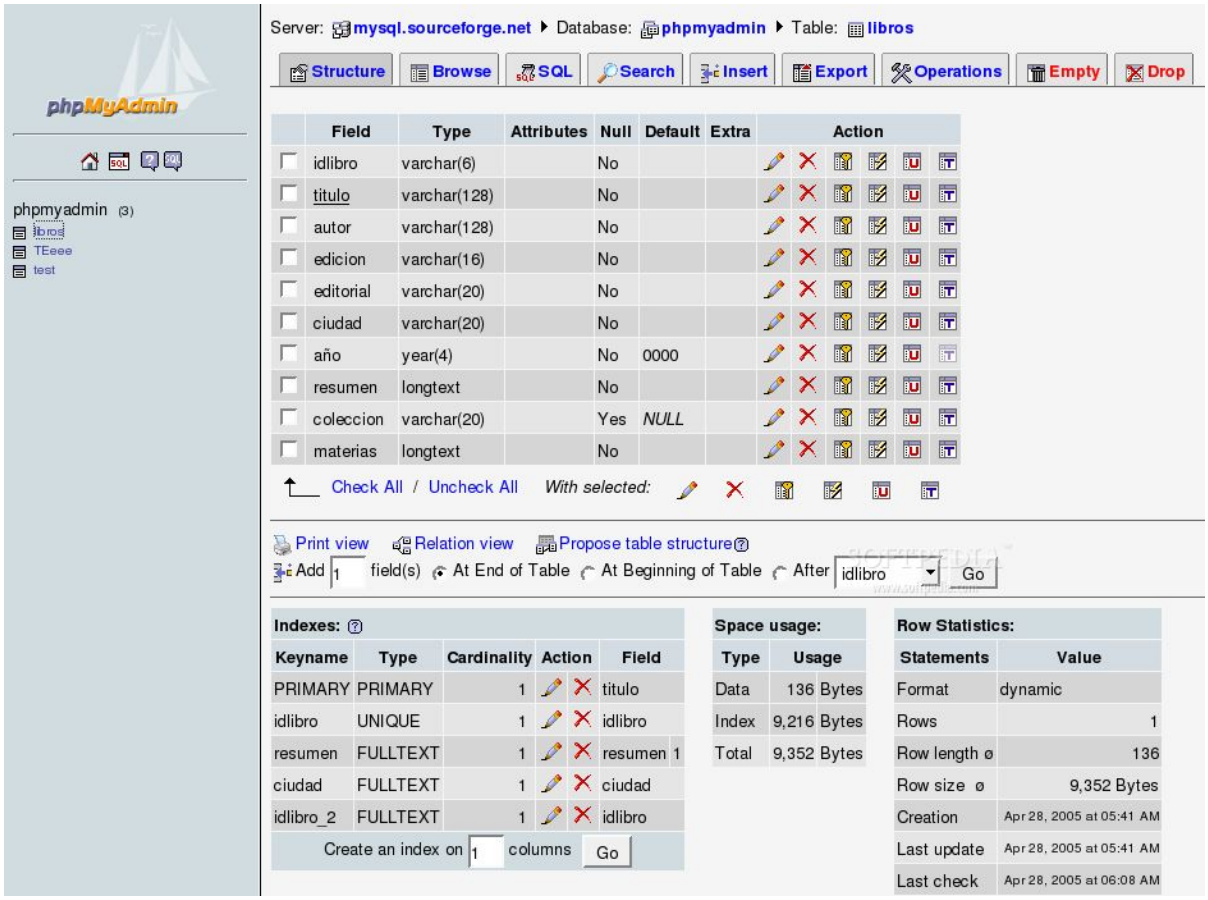
3.2 Webové aplikace

Systém Caché nabízí webovou aplikaci Management Portal, která umožňuje spravovat nastavení systému. Její součástí ale není rozhraní pro přidávání nových entit, ani editaci jejich zdrojového kódu. Zaměření aplikace Management Portal je tedy odlišné a pro inspiraci k vytvoření webového nástroje pro editaci entit není postačující; rozhodl jsem se tedy inspirovat se možnostmi aplikací s podobným zaměřením, jako je zadání této práce.

Pro editaci zdrojového kódu by byl vhodný nějaký javascriptový framework, tedy nástroj, který by graficky zvýraznil syntaxi zdrojového kódu. Pokusím se jej tedy najít.

3.2.1 Webové aplikace podobného zaměření

Vybral jsem jedinou, svým zaměřením nejpodobnější, webovou aplikaci *phpMyAdmin*.



Obrázek 2: phpMyAdmin

Tato aplikace je určena pro správu relační databáze. Po levé straně si můžeme všimnout přehledu jednotlivých databází. V horním úseku obrazovky jsou přehledně vidět možnosti editace prvků a nastavení. PhpMyAdmin se zaměřuje především na funkčnost a jednoduchost použití uživatelského prostředí.

3.2.2 Javascriptový framework pro editaci zdrojového kódu

Javascriptové frameworky určené ke grafickému zvýraznění syntaxe existují pro většinu rozšířených programovacích jazyků. Pro soubor jazyků, se kterými pracuje systém Caché, se mi ovšem žádný veřejně dostupný framework nepodařilo najít.

4 Entity v Caché (vnitřní reprezentace dat)

Pro vytvoření webového nástroje potřebujeme znát typy entit, se kterými budeme pracovat, dále jejich dostupné vlastnosti a vzájemné provázání. Každá entita je určena skrze svou třídu, která ji popisuje.

Pro lepší porozumění je potřebné vymezit si význam pojmu *třída*. Tento pojem se zde vyskytuje ve dvou významech. První představuje *předpis* pro objekty v objektovém programování a druhý představuje samotnou *entitu* třída v systému Caché (podobně jako např. entita rutina). Oba významy budou odlišeny kontextem, případně konkrétnějším popisem.

V této kapitole zpracovávám typy entit, jejich hierarchii a možnosti přístupu. Součástí kapitoly je i ukázka zdrojového kódu pro operace s třídami a rutinami. Tento kód je využitím vestavěného API systému Caché. (Více podrobností v kapitole 2 a 8.)

Důležitým tématem jsou také jmenné prostory, které ovlivňují dostupnost tříd a jejich vztahů. Dozvíme se důvody, proč je nutné jmenné prostory dynamicky měnit při běhu aplikace.

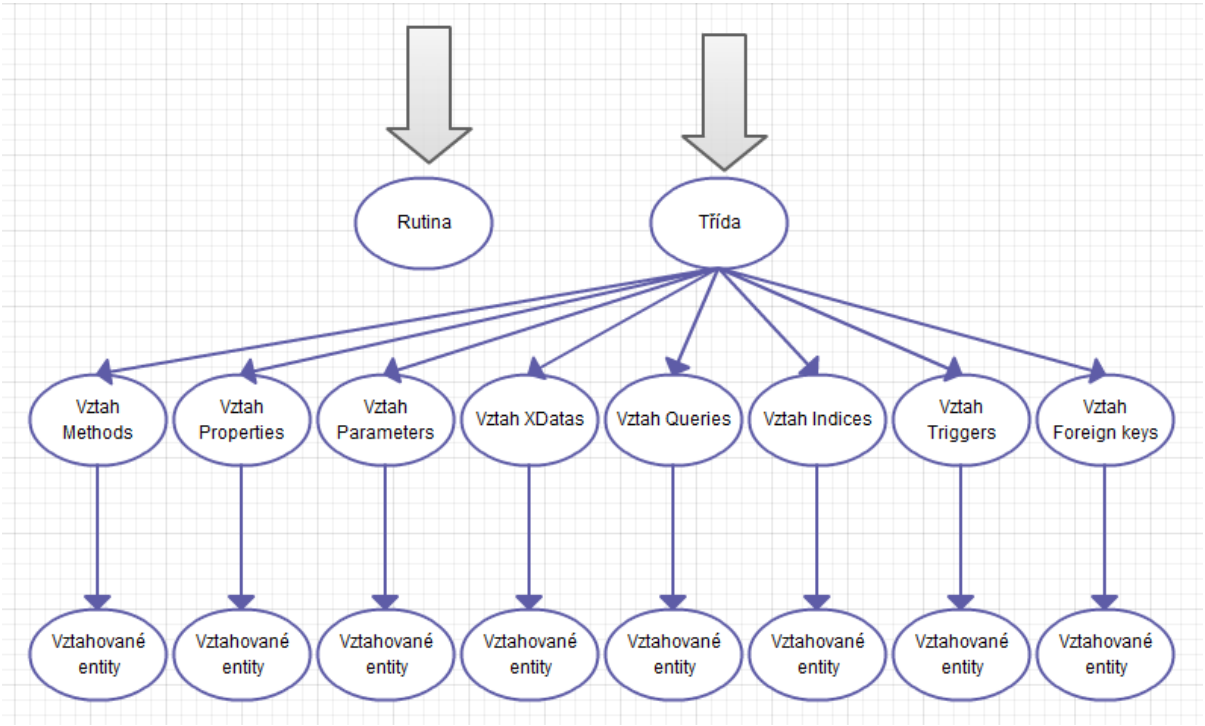
Informace obsažené v této kapitole jsem čerpal ze zdokumentovaného API systému Caché[3] a z vlastního zkoumání systému.

4.1 Typy entit v caché

Systém Caché obsahuje mnoho typů entit. Jejich plný výčet je dostupný přes API Caché a jeho dokumentaci. Pro zjednodušení zde popíšu pouze takové entity, které jsou nutné pro znázornění vnitřní reprezentace dat rutin a tříd.

4.1.1 Hierarchie entit

Jak vidíme z obrázku 3, k vytvoření systému postačuje detailní znalost entit rutina a třída a jejích vnořených entit. Takto vnořené entity jsou dostupné přes entitu vztah v rámci objektu třída. Vztah poté může obsahovat libovolný počet vztahovaných entit. Typickou vztahovanou entitou je například metoda třídy. Bližšímu popisu entitě vztah se věnuje kapitola 4.1.4.



Obrázek 3: Hierarchie entit

4.1.2 Entita rutina

Entita rutina se liší od třídy tím, že neobsahuje žádnou entitu typu vztah. Rutina obsahuje pouze zdrojový kód a žádné jiné vlastnosti.

4.1.3 Entita třída

Oproti rutině je třída již značně komplexnější datový typ. Entita třída obsahuje mnoho vlastností týkajících se samotné entity, které popisují a mění její chování.

Kromě těchto vlastností obsahuje entita třída další entity typu *vztah*, který popíšu v následující podkapitole.

4.1.4 Entita vztah v rámci entity třída

Entitu vztah si lze představit jako homogenní dynamické pole odkazů na další objekty. Entita třída obsahuje 8 typů vztahů (rozděleno dle typu vztahovaného objektu). Jednotlivé typy vztahů poté mohou obsahovat 0 – N vztahovaných entit.

4.1.4.1 Typy vztahů s entitou třída

Výčet vztahů, které entita třída může obsahovat, můžeme vidět níže. Pro lepší přehlednost ponechávám pojmy v angličtině s českým překladem v závorce.

- Methods (metody)
- XDatas (xdata)
- Queries (dotazy)
- Properties (atributy)
- Indices (indexy)
- Parameters (parametry)
- Foreign keys (cizí klíče)
- Triggers (triggery)

Každá z těchto vztahovaných entit obsahuje svoje vlastnosti. Ne ke všem ovšem budeme chtít ve vytvářené aplikaci povolit přístup (např. vlastnost „parent“, která odkazuje na druhý objekt vztahu – tedy entitu třída). Bude tedy potřebné provést určitou selekci vlastností entity kvůli výpisu a možné editaci.

4.2 Přístup k entitám

Každá z entit obsahuje jiné typy dat. Entity lze rozdělit na dvě skupiny: první tvoří entity třída a rutina a druhou tvoří entita vztah se vztahovanými objekty. Entity v rámci takto rozdělených skupin spojuje (ne) jedinečnost jmen a právě rozdíly v přístupu k těmto entitám.

4.2.1 Přístup k entitám rutina a třída a jejich vlastnostem

Název těchto entit je v rámci jmenného prostoru jedinečný, lze tedy pomocí něj entitu jednoznačně identifikovat. Otevření objektu s definicí rutiny, resp. třídy lze provést přímo, tedy bez potřeby otevření nějakého objektu, který by sloužil jako prostředník. Načtení provedeme následovně:

```
set tRoutineDef = ##class(%Routine).%OpenId(RoutineName)
set tClassDef = ##class(%Dictionary.ClassDefinition).%OpenId(ClassName)
```

V objektu `tRoutineDef` (resp. `tClassDef`) nyní máme dostupné všechny vlastnosti rutiny, resp. třídy.

4.2.2 Přístup k objektům vztahu a jejich vlastnostem

Název těchto entit není jedinečný v rámci jmenného prostoru, ale pouze v rámci vztahované třídy. K těmto entitám ani nelze přistoupit jako k samostatnému objektu. Pro načtení vztahů je tedy nutné načíst objekt s definicí třídy jako prostředníka (viz. minulá kapitola 4.2.1.). Tento objekt poté obsahuje 8 vlastností typu vztah (výčet v kapitole 4.1.4.), což je homogenní dynamické pole o počtu 0-N položek, jak již bylo popsáno výše. K těmto vztahovaným entitám lze přistupovat pomocí jména třídy, typu vztahované entity a jedinečného názvu entity.

```
set methods = tClassDef.Methods
for i = 0:1:tClassDef.Count()-1
{
      s method = tClassDef.GetNext(i)
      if (WantedMethodName = method.Name)
      {
            // objekt method nyní obsahuje vlastnosti hledané metody
      }
}
```

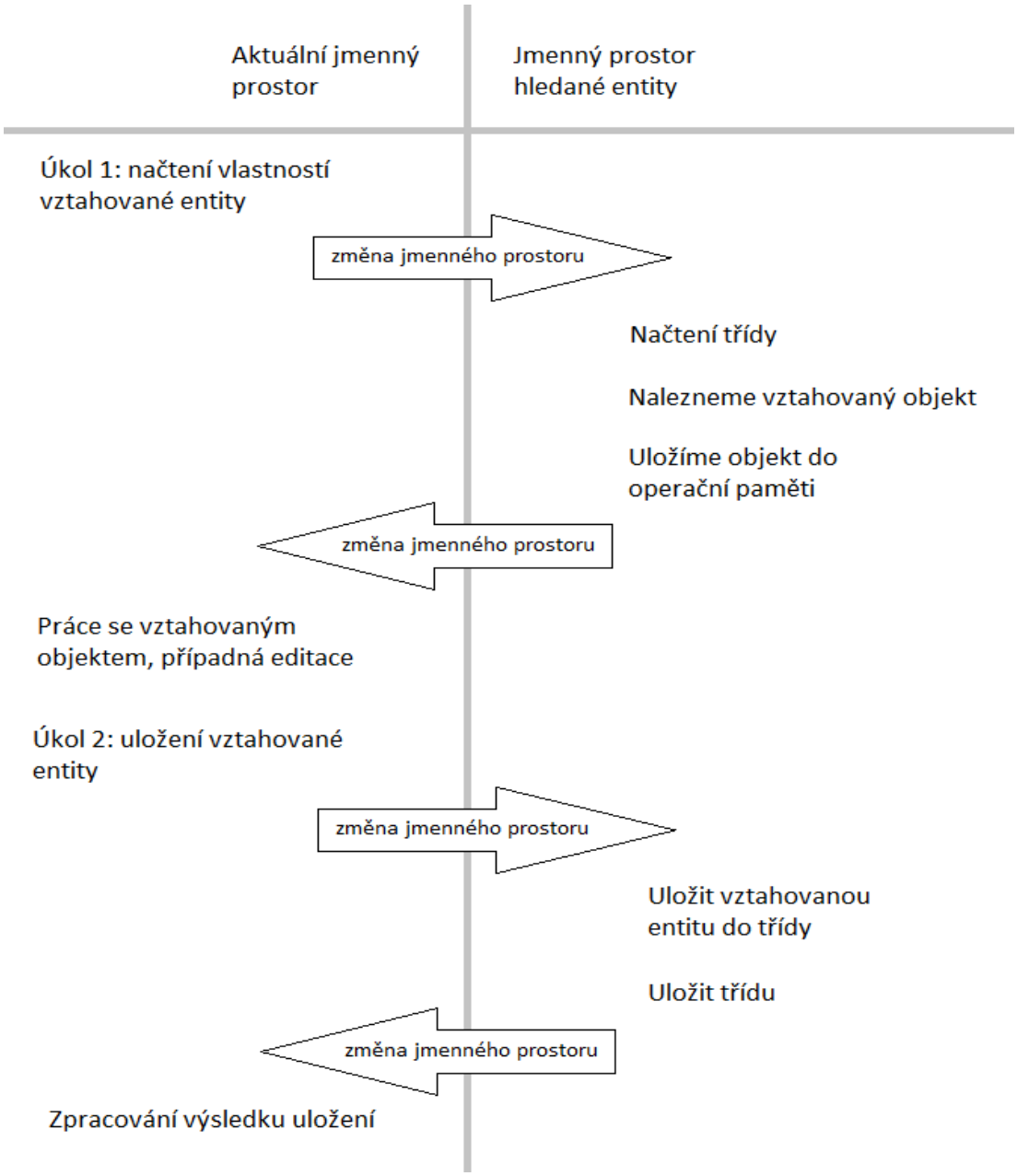
4.2.3 Provázání entit se jmennými prostory

Pro korektní načtení a uložení jakékoliv entity, která není součástí aktuálního jmenného prostoru, je nutné tento jmenný prostor změnit na takový, ve kterém se nachází právě hledaná entita. Po načtení hledané entity je třeba jmenný prostor změnit zpět na výchozí. Lze předpokládat, že vzhledem k zaměření vytvářené aplikace k tomuto jevu bude docházet poměrně často.

Pro objekty typu vztah nestačí pouze načíst definici třídy a změnit jmenný prostor zpět. Entita vztah je odkaz, který se při změně jmenného prostoru znepřístupní. Pro správnou operabilitu je nezbytné, aby bylo provedeno uložení hledané vztahované entity do operační paměti bezprostředně po načtení definice entity třída. Poté může být jmenný prostor změněn zpět a lze pokračovat ve zpracování dané vztahované entity.

Neprovede-li se změna jmenného prostoru zpět, zpracování skončí chybou kvůli neexistenci tříd a metod vytvořeného webového nástroje, protože nejsou součástí nyní aktuálního jmenného prostoru.

Uložení vztahů poté probíhá analogicky. Průběh těchto procesů můžeme vidět na obrázku 4.



Obrázek 4: dynamické měnění aktuálního jmenného prostoru

4.3 Vlastnosti entit

Každá entita vyskytující se v systému Caché (vztahovaná, či nikoliv) má svoje vlastnosti. Pro příklad si uvedeme několik typických vlastností entit (pro přehlednost v angličtině):

- Name
- Final
- Abstract

Vlastností entit existuje mnoho a jsou pro každou entitu speciální. Po podrobnějším prozkoumání systému jsem zjistil, že je lze rozdělit do tří skupin podle povolených hodnot, kterých mohou nabýt:

- vlastnost s hodnotou typu boolean
- vlastnost s hodnotou typu „výběr ze seznamu“
- vlastnost s hodnotou typu „libovolná hodnota“

4.4 Zdrojový kód entit

Zdrojový kód se nevyskytuje u všech entit. Typickým zástupcem entity se zdrojovým kódem je například metoda či rutina. Zdrojový kód je ve své podstatě také vlastností, ale s tak odlišným přístupem, že jsem se rozhodl jej pro další zpracování ze skupiny vlastností vyřadit. Bližší důvody a detaily jsou popsány v následující podkapitole a v kapitole 8, protože se již týkají vlastní implementace.

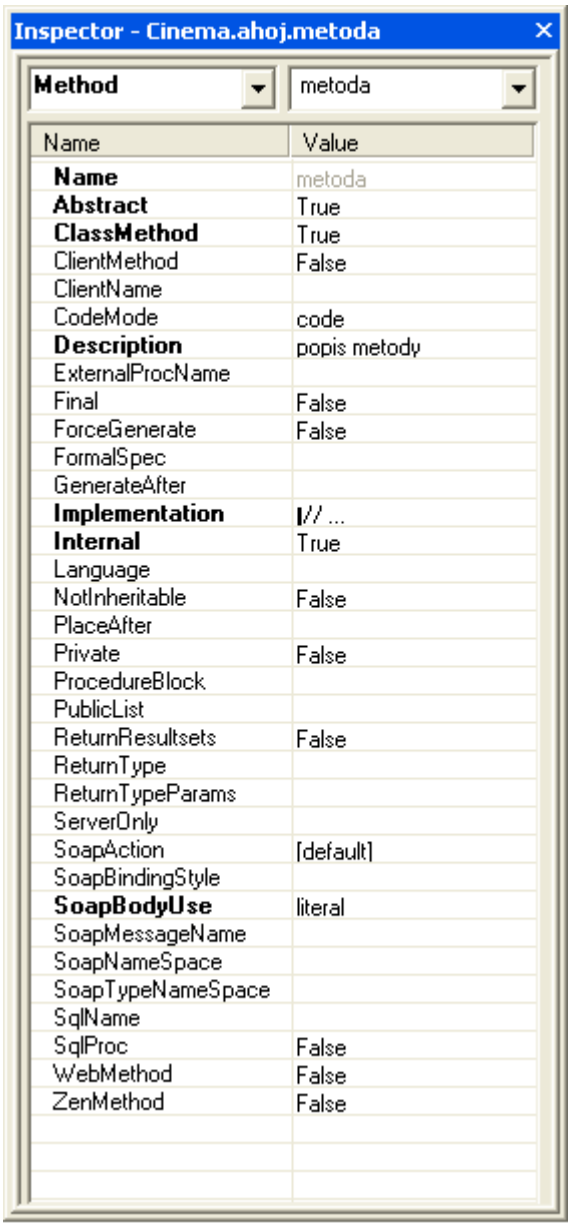
4.5 Editace vlastností entit v nástroji Caché Studio

Nyní, když detailněji známe zákonitosti entit a seznámili jsme se i s Caché Studiem, se můžeme blíže podívat na způsob, jakým Caché Studio pracuje se svými entitami. Vnitřní zdrojový kód metod, dotazů (query) a podobných vztahovaných entit nás v tuto chvíli nezajímá, proto jej z této analýzy vypustíme. Naopak se zaměříme na zpracování vlastností entit.

Vlastnosti entit můžeme v Caché Studiu editovat dvěma způsoby – přímou editací zdrojového kódu (obrázek 5), nebo editací v přehledu vlastností entity (obrázek 6).

```
/// popis metody
ClassMethod metoda() [ Abstract, Internal, SoapBodyUse = literal ]
{
    // ...
}
```

Obrázek 5: Vlastnosti metody reprezentované zápisem ve zdrojovém kódu



Obrázek 6: Vlastnosti metody reprezentované přehledem

Znalost oddělení entity třída od svých vztahovaných entit a fakt, že výše uvedené způsoby editace jsou naprostro rozdílné, nám dávají jistou indicii, jak probíhá zobrazení a uložení entity třída spolu se všemi svými vztahovanými entitami v podobě uceleného zdrojového kódu v Caché Studiu.

Jako příklad si uvedeme načtení třídy, která obsahuje jednu metodu a jeden parameter.

Caché Studio nejdříve načte hledanou třídu i s jejími vlastnostmi. Tyto vlastnosti vypíše stejným stylem, jaký vidíme na obrázku 5. Poté načte všechny vztahy, které vedou ke vztahovaným entitám. Tímto způsobem objeví onu metodu a opět načte a zobrazí všechny vlastnosti (tentokrát ty,

které patří té konkrétní metodě) a poté zobrazí vnitřní zdrojový kód metody. Více metod již třída neobsahuje, přejde se proto na další vztah a podobným způsobem je zpracován i onen parametr třídy. Další vztahy třída neobsahuje, výpis třídy se tedy ukončí.

Naproti tomu editace a následné uložení vyžaduje důležitý krok od Caché Studia – kontrolu hodnot vlastností (kvůli tomu, že hodnoty vlastností lze vkládat i ručně). Jak již víme z kapitoly 4.3., ne všechny entity totiž mohou obsahovat libovolné hodnoty. Takto vzniklý konflikt je nahlášen jako chyba při překladu.

Toto chování lze brát jako důkaz, že editace vlastností provedená přímo ve zdrojovém kódu se následně převede do souboru vlastností uvedených v přehledu, jak je uvedeno na obrázku 6, a tedy uloží se do vnitřní reprezentace dat entity.

5 Analýza požadavků

V této kapitole se zaměřím na vymezení cílů aplikace a na to, jaké nejdůležitější funkce by měla obsahovat. Při určení hlavních cílů vycházím především ze zadání práce, proběhlých konzultací, analýzy dostupných nástrojů (včetně API Caché) a vlastních zkušeností, které vyplynuly z využívání desktopové aplikace Caché Studio.

Nejdříve začnu popisem základního principu, kde podrobněji zpracuji zadání a rozšířím jej o další cíle. Poté se dostanu ke zpracování systémových požadavků a instalaci vytvořené aplikace. Dostupné akce uživatele aplikace znázorním pomocí use-case diagramu a poté vymezím požadavky na bezpečnost aplikace. Pro návrh aplikace a její potencionální úspěšnost je nutné zanalyzovat i zamýšleného koncového uživatele. Se znalostí typického koncového uživatele můžeme odhadnout jeho požadavky na aplikaci.

5.1 Základní princip

Jak již bylo zmíněno v kapitole 1, cílem této práce je vytvořit webovou aplikaci pro editaci a vytváření nových entit v systému Caché. Tato aplikace umožní uživatelům editovat zdrojový kód a vlastnosti entit. Jádro aplikace bude využívat API Caché pro práci s těmito entitami. Nad tímto jádrem bude vystavěno uživatelské prostředí, které uživateli zprostředkuje všechny potřebné akce.

Aplikace bude vytvořená v systému Caché s využitím jazyka Objectscript [1]. Všechna chybová hlášení musí aplikace zpracovat a zobrazit vlastní hlášení. Například stane-li se, že se uživatel pokusí přistoupit na neexistující třídu, nebo třídu, k níž nemá přístup, aplikace jej na to upozorní srozumitelným textem.

Vytvořená aplikace musí umožnit změnu jmenného prostoru, aby bylo možné uživateli zpřístupnit všechny entity, které má právo vidět. V zadání bakalářské práce je uveden požadavek na editaci rutin, tříd a metod. Tento požadavek se pokusím rozšířit na editování všech typů vztahovaných entit.

V rámci jmenného prostoru je nutné zobrazit rutiny, třídy a vztahované entity vypsané v kapitole 4. Aplikace umožní každé zobrazené entitě editaci jejích vlastností. Některé entity v sobě kromě vlastností ponesou i zdrojový kód.

Při editaci zdrojového kódu bude třeba zajistit možnost vložení a zobrazení *všech možných znaků a sekvencí* v oblasti, která k tomu bude určena. Ideální by byla existence javascriptového frameworku pro grafické zobrazení zdrojového kódu pro všechny možné programovací jazyky, které v sobě systém Caché nosí. Bohužel, z analýzy v kapitole 2 již víme, že takový nástroj zatím neexistuje. Tvorba tohoto nástroje by byla značně složitá a časově by přesahovala možnosti této

práce. Pro zajištění plné funkčnosti vytvářené aplikace bude ovšem nutné implementovat alespoň základní potřeby.

Aplikace dle zadání musí umožňovat také vytváření nových entit v rámci zvoleného jmenného prostoru.

Abychom mohli provedené změny zpětně promítnout do databáze, každá entita musí mít možnost uložení a kompilace. Výsledek těchto operací musí být zviditelněn uživateli.

Aplikace dále nabídne možnost bezpečného odhlášení po ukončení práce.

Z konzultací vyplynulo, že pro možné rozšíření aplikace po veřejnosti by bylo vhodné implementovat možnost překladů tohoto nástroje. Všechny hlášky zobrazované uživateli tedy budou přeložené do vybraného jazyka.

Pro ověření funkčnosti vytvořené aplikace jsme na konzultacích zvolili metodu, při které bude vytvořena jednoduchá webová aplikace v systému Caché pouze za použití vytvářeného nástroje.

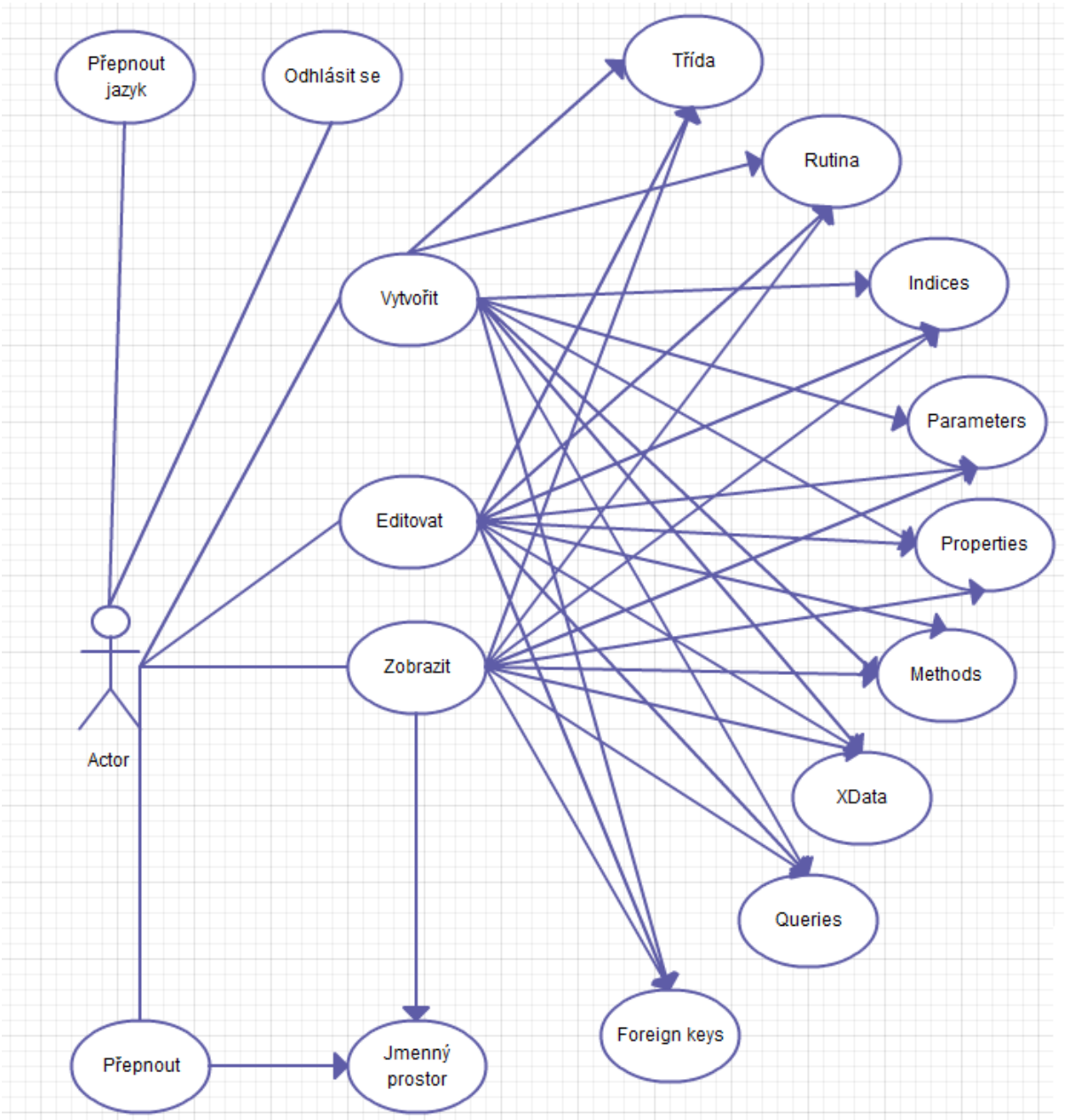
5.2 Instalace aplikace

Instalace aplikace musí být co nejjednodušší. Systém Caché má v sobě implementovaný export a import tříd [2], vytvořená aplikace tedy tento systém využije ke svojí instalaci. Ideální by bylo, kdyby se instalace prováděla pouze pomocí importu takto vyexportovaných balíčků.

Dalším požadavkem na instalaci a spuštění vytvořené aplikace je samostatnost (stand-alone). Tento přístup zaručí, že aplikace nebude vyžadovat další přídavný software, případně bude přibalen v instalačním balíčku.

5.3 Use case diagram

Všechny akce uživatele jsou nyní známy, je tedy možné specifikovat use case diagram už v této fázi. Vztahované entity jsou opět uvedeny v angličtině.



5.4 Omezení přístupu

Při vstupu do aplikace musí být požadováno přihlášení. Jedná se aplikaci, která může vést k omezení nebo ztrátě funkčnosti webových aplikací, výraznému omezení bezpečnosti, či vynášení citlivých dat. Přihlašování je tedy bezpodmínečně nutné.

Samotné přihlášení ovšem nestačí, bude nutné respektovat i uživatelské role v systému. Bylo by nesmyslné, aby účet s rolí „Uživatel“ mohl nahlédnout do tříd a metod, nebo dokonce měnit funkčnost webové aplikace.

Tímto bychom měli specifikované požadavky na autentizaci a autorizaci uživatele. Ideální by bylo, kdyby se podařilo správu uživatelů kompletně přenechat na systému Caché. Caché by potom obstaralo správu těchto uživatelů i s přiřazením jejich rolí a vytvořená aplikace by pouze převzala možnosti, které jsou dostupné právě přihlášenému uživateli.

5.5 Cílový uživatel a bližší zaměření aplikace

Z podstaty zadání vyplývá, že vytvářená webová aplikace *nebude* masově veřejně přístupná, právě naopak – bude vyžadovat přihlášení (jak již víme z předchozí kapitoly) a nebude umožňovat registraci do systému. Díky svému zaměření a volbám, které bude aplikace umožňovat, lze odhadnout vlastnosti cílového uživatele:

- Uživatel bude znát zákonitosti, vlastnosti a chování systému.
- Uživatel má přímý přístup k databázi (jinak by se ani nemohl do aplikace přihlásit).
- Uživatel bude aplikaci používat k práci.
- Uživatel má před sebou jasný úkol, který chce splnit.

Z těchto znalostí vyplývá, že typický cílový uživatel bude mít nějakou úroveň správce systému. Můžeme přejít k analýze toho, co bude tento zamýšlený uživatel od aplikace vyžadovat:

- rychlé splnění svého úkolu
- přehlednost a snadnou orientaci
- funkčnost

Tyto odhady nám pomohou při návrhu uživatelského rozhraní.

6 Návrh systému

V této fázi jsme již seznámeni se systémem Caché a jeho důležitými částmi pro tvorbu této práce. Známe podrobné zadání, požadované akce uživatele a můžeme tedy přejít k samotnému návrhu systému.

Z kapitoly 2 víme, že jádro aplikace bude pro načtení a editaci entit využívat API systému Caché. Nejdříve určíme přesnou podobu zobrazení entit. Díky této znalosti budeme moci zpracovat návrh uživatelského prostředí, poté přejdeme k určení technologií, které nám danou webovou aplikaci umožní vytvořit. Dále navrhnu architekturu a strukturu aplikace, zamyslíme se nad potřebami konfigurace aplikace a realizací překládání uživatelských hlášení.

6.1 Zobrazení a možnosti editace vztahovaných entit a entity třída

Pro další pokračování v návrhu aplikace je nutné ujasnit si přístup k zobrazování entity třída a jejích vztahovaných entit. Již víme, že Caché Studio před zobrazením entity třída načte její vlastnosti i všechny její vztahované entity (a jejich vlastnosti) a tyto entity zobrazí jako celek v podobě zdrojového kódu entity třída.

Zanalyzujme si nyní tuto metodiku a vlastnosti Caché Studia:

- Při uložení a následné kompilaci je nutné přiřadit hodnoty vlastností korespondujícím vlastnostem a validovat je – důsledkem je nutnost implementace lexikálního, syntaktického a sémantického analyzátoru pro kontrolu hodnot těchto vlastností.
- Caché Studio má funkční rozhraní pro vrácení změn a poskytuje jednoduchou dynamickou kontrolu syntaxe. Caché Studio tedy není tak náchylné na chyby při překladu vzniklé z nepozornosti či překlepu, případně lze změny jednoduše vrátit zpět.

Vytvoření takového analyzátoru by bylo příliš pracné a ani není tématem této práce. Zobrazení třídy jako celek (vlastnosti třídy a s nimi i všechny její vztahované entity s jejich vlastnostmi) by navíc mohlo vést ke ztrátě přehlednosti a orientace (webová aplikace nebude umožňovat vrácení změn, a dále kvůli neexistenci javascriptového frameworku pro kontrolu syntaxe a podbarvení zdrojového kódu).

Z těchto důvodů jsem se rozhodl zvolit jiný přístup: aktivní bude vždy jen jedna entita a její vlastnosti se zobrazí výčtem, nikoli jako součást zdrojového kódu. O entitách víme, že rutina a třída jsou jedinečné v rámci jmenného prostoru a vztahované entity jsou jedinečné v rámci entity třída. Řešením navigace mezi nimi by tedy bylo vytvořit *dva* navigační panely, přičemž jeden by zajišťoval

navigaci jedinečných entit v rámci jmenného prostoru a druhý by zajišťoval navigaci pro jedinečné entity v rámci entity třída. Tímto krokem tedy oddělíme entitu třída od jejích vztahovaných entit; aktivní (zobrazená) bude vždy jen jedna entita (třída, rutina nebo vztahovaná entita). Důsledky jsou takovéto:

- Odpadá nutnost zpracovávat vlastnosti entity ze zápisu zdrojového kódu – lze je přehledně zobrazit a měnit výčtovými prvky (např. html prvkem <select>).
- Zlepší se orientace v aplikaci – uživatel bude vždy přesně vědět, v jaké entitě se nachází.

S těmito důsledky se mi tento přístup jeví jako správný. Omezíme výskyt chyb, respektive zpřesníme jejich lokalizaci – při editaci entity a následné chybě je jasné, že chyba nastává právě kvůli editované entitě a zlepšíme orientaci v aplikaci.

Další možný přístup by byl kombinací obou předchozích. Entita třída by se zobrazila spolu se všemi jejími vztahovanými entitami, ale vlastnosti všech entit by se zobrazily výčtovými prvky, tedy ne v podobě zdrojového kódu. Tímto přístupem bychom ale ztratili onu výhodu ve zlepšení orientace.

Výsledkem tohoto zamyšlení je tedy skutečnost, že v aplikaci musí být dva navigační prvky a aktivní bude vždy pouze jedna entita. Její vlastnosti se budou vypisovat výčtovými prvky a nikoli v podobě zdrojového kódu, jak je tomu v Caché Studiu.

6.2 Výběr vlastností pro entity

Jak již byl zmíněno, každá entita má svoji vlastní sadu vlastností. Tyto vlastnosti lze vypisovat dvěma způsoby – automaticky vypsat všechny, nebo ručně zvolit vlastnosti k výpisu. Vzhledem k tomu, že některé vlastnosti jsou interního charakteru a žádný uživatel nemá právo na jejich změnu, rozhodl jsem se pro manuální určení vlastností k výpisu a editaci pro každou entitu zvlášť. Hodnota vlastností, které nebudou zobrazeny pro editaci, zůstane nezměněna.

6.3 Návrh uživatelského rozhraní

Na začátku provedu vyhodnocení analýzy existujicích editorů, kde vyzdvihnu jejich silné a slabé stránky a zhodnotím možnosti inspirace. Se znalostí závěru tohoto vyhodnocení provedu návrh webové aplikace a návrh popisu. Zaměřím se i na navržené funkční prvky uživatelského rozhraní. Poté popíšu jednotlivé typy zobrazovaných stránek a popíšu navigaci v aplikaci.

6.3.1 Shrnutí analýzy požadavků a možnosti inspirace

V kapitole 3 jsme provedli analýzu možností a uživatelských rozhraní různých editorů (desktopových i webových). Oba tyto editory mají své přednosti a rád bych se jimi při návrhu uživatelského rozhraní inspiroval. Nejprve si v krátkosti shrňme, jaké možnosti vytvářená aplikace musí nabízet:

- výpis a změnu jmenného prostoru
- výpis entit (a navigaci mezi nimi)
- zobrazení aktivní entity a jejích vlastností
- tvorbu nových entit
- možnost nastavení aplikaci (překlady, …)
- odhlášení.

Z kapitoly 5.6 víme, že aplikace bude zaměřena především na *jednoduchost použití*, *snadnou orientaci* a *funkčnost*. Zde vidíme podobnost se zpracováním webové aplikace phpMyAdmin [4]. Právě tímto jejím přístupem a rozvržením prvků se budu inspirovat i při tvorbě návrhu:

- navigační prvky u kraje obrazovky
- možnosti aplikace v horní části obrazovky
- obsah editovaného objektu v hlavní části obrazovky.

Inspirace v desktopové aplikaci Caché Studio přidá k návrhu tyto prvky:

- spodní lišta s informacemi o výsledcích prováděné akce
- možnost nastavení velikosti prvků prostředí
- upřesnění možností navigace
- zobrazení a editace vlastností entity bude provedena výčtem těchto vlastností (jako je popsáno na obrázku 6)

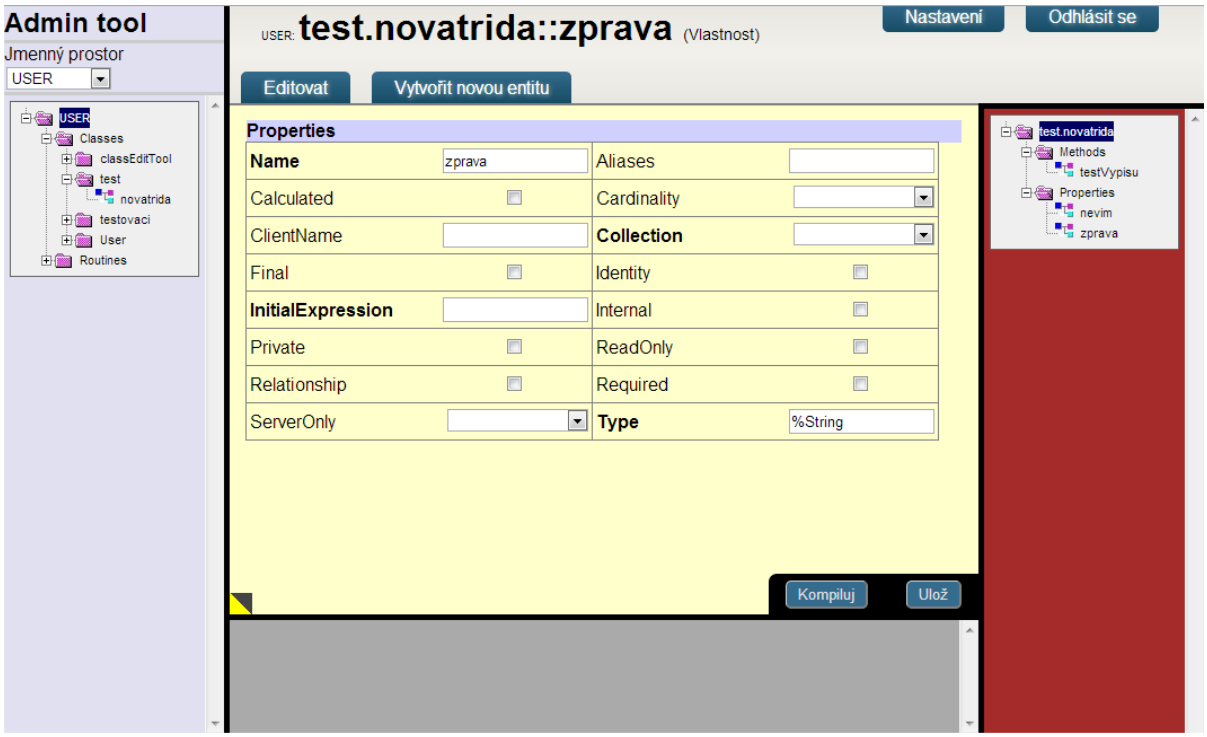
6.3.2 Popis uživatelského prostředí a jeho funkčností

Sjednocením znalostí nabytých v minulé podkapitole můžeme vytvořit popis návrhu vytvářené aplikace:

- Navigační prvky ve sloupcích po stranách obrazovky – na levé straně budou rutiny a třídy, na pravé straně budou vztahované entity v rámci třídy.
- V horním pásu obrazovky bude ovládání aplikace.
- Hlavní obsah aplikace bude uprostřed obrazovky.
- Ve spodní liště budou informace s výsledkem ukládání a kompilace.
- Prvkům uvedeným výše půjde nastavit velikost.
- Do návrhu zakomponujeme ovládací prvky na uložení a kompilaci.

6.3.3 Návrh rozhraní

V této chvíli již přesně víme, jaké vlastnosti má uživatelské prostředí mít a jaké volby má umožňovat, můžeme se tedy pustit do samotného grafického návrhu.



Obrázek 7: Grafický návrh webové aplikace

6.3.4 Hlavní obsah aplikace

Hlavní obsah se bude zobrazovat ve středové části aplikace a bude závislý na zvolené akci. Při aktivní entitě zobrazí její vlastnosti a případně zdrojový kód (u metod, rutin, apod., u kterého bude nutné zajistit vkládání tabelátoru a escapovat sekvence – blíže kapitola 7), při akci „vytvoř novou entitu“ nabídne formulář pro tvorbu nové entity a při akci „nastavení“ zobrazí možnosti konfigurace aplikace.

6.3.5 Navigace

Při návrhu zobrazení navigačních prvků po stranách aplikace jsem se inspiroval aplikací Caché Studio. Hierarchie entit po levé i pravé straně bude mít podobu stromu.

Kořenem stromu po levé straně bude název jmenného prostoru, který se bude dělit na rutiny a třídy, resp. balíčky tříd, které se poté budou dále dělit na jednotlivé třídy.

Kořen stromu po pravé straně bude tvořit název třídy. Ten se bude dělit na prvky, které představují vztahované entity, jsou-li přítomné.

V horní části aplikace bude jasně uvedeno, která entita je právě editována. Pro vztahované entity bude uvedeno i jméno třídy, ke které se entita vztahuje. Bude-li aktivní jiný typ stránky, než je editace entity (například stránka nastavení), její nadpis se zobrazí právě zde.

6.3.6 Tvorba nových entit

Vztahovanou entitu nelze vytvořit bez toho, abychom věděli do které třídy má patřit. Tvorba tohoto typu entit bude tedy podmíněna zvolenou třídou.

Aplikace dále musí kontrolovat jedinečnost jmen vytvářených entit a o případném konfliktu informovat uživatele.

6.4 Výběr technologií

S hotovým návrhem uživatelského rozhraní mohu přejít k výběru technologií. Aplikační vrstva je jasně dané zadáním – využiji systém Caché a jeho skriptovací jazyk ObjectSript [1].

Další požadavek zadání – tedy webové rozhraní – určuje použití jazyka HTML spolu s přidruženými technologiemi CSS a Javascript.

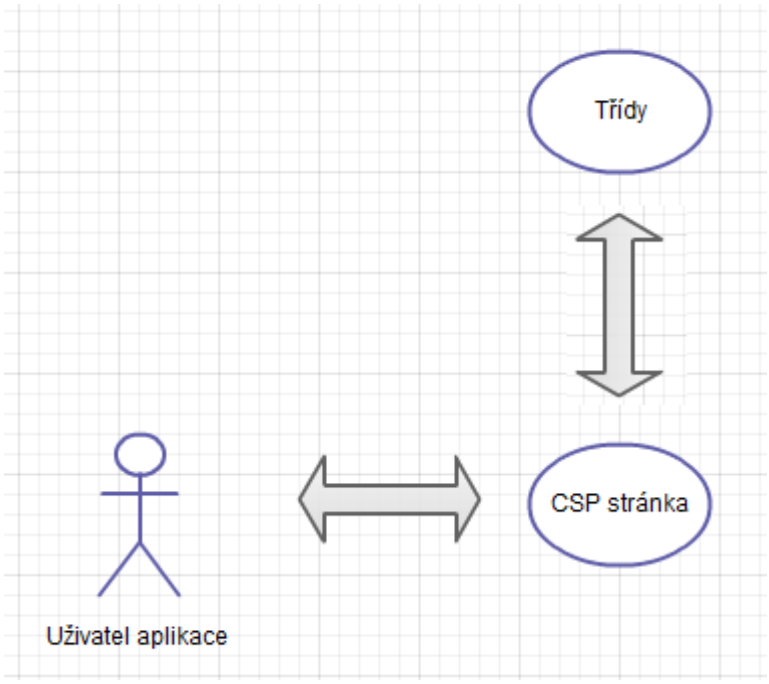
Jazyky HTML a CSS jsou v současné době u webových aplikací samozřejmostí a proto se jejich popisu nebudu v této práci věnovat. Aplikace nebude obsahovat prvky z připravovaného standardu HTML 5, ani nebude využívat nových možností přidaných v CSS 3 (alespoň ne u prvků, které jsou zásadní pro ovládání aplikace).

Pro zobrazení a práci s navigačními prvky v podobě stromů jsem se rozhodl využít javascriptového frameworku. Na výběr je jich několik, já jsem zvolil framework MooTools [5] s knihovnou Mif.Tree [6], která poskytuje kvalitní řešení stromového menu. Zdrojem dat pro tato menu bude JSON objekt [7].

Pro uchování velikostí prvků v uživatelském rozhraní využiji cookies, které se nastaví při každé změně velikosti uživatelského rozhraní. Při dalším načtení tedy velikosti setrvají, což je i cílem této funkčnosti.

6.5 Struktura aplikace

Se zvolenými technologiemi můžeme určit strukturu aplikace.

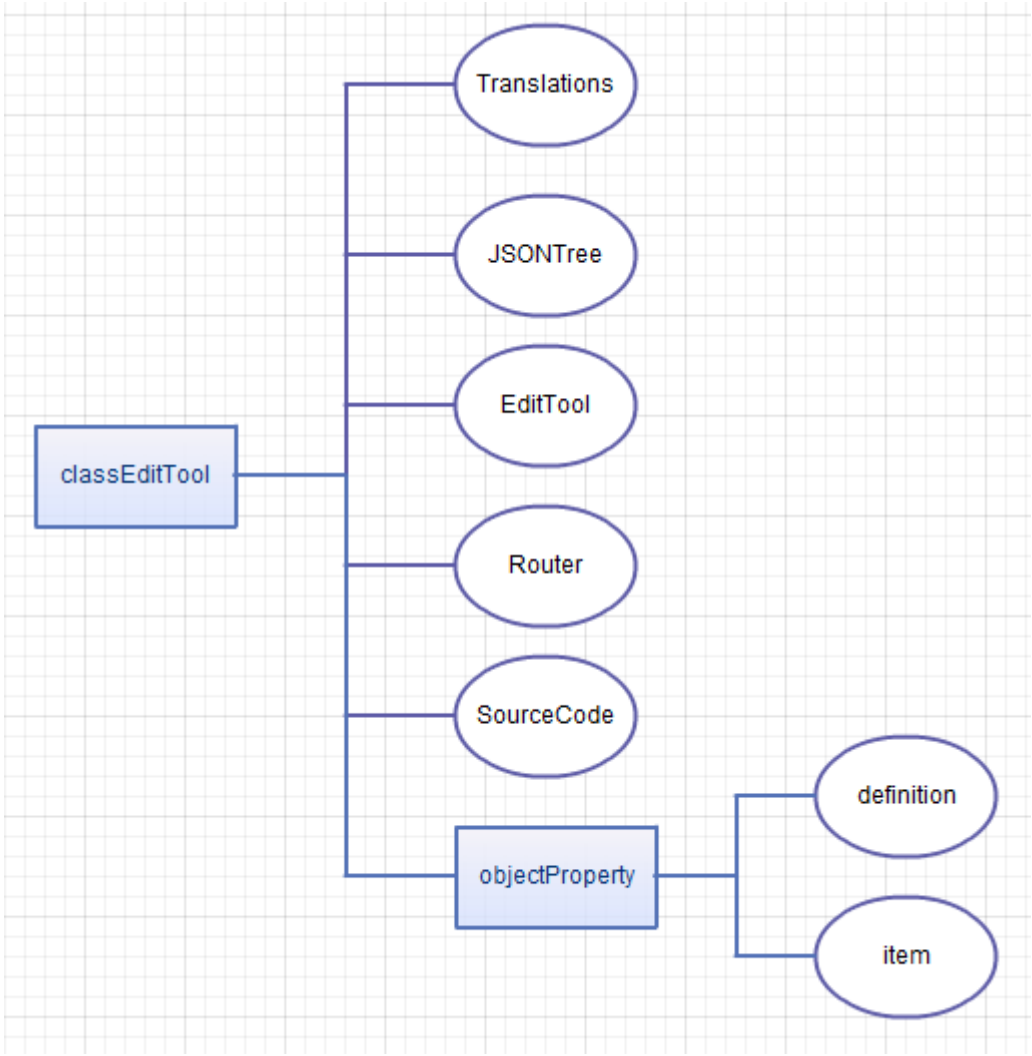


Obrázek 8: Struktura aplikace

Jak vidíme z obrázku 8, uživatel bude s aplikací komunikovat skrze CSP stránku. V této podkapitole blíže probereme uvedený přístup a popíšeme jednotlivé složky.

6.5.1 Diagram tříd

Všechny třídy, které bude aplikace pro svůj běh využívat, budou součástí balíčku classEditTool, který pro tento účel vytvořím.



Obrázek 9: Diagram tříd

Na obrázku 9 můžeme vidět navrhnutý diagram tříd. Tyto třídy blíže popíšu.

Translations – tato třída bude obstarávat překlady. Z této třídy bude vytvořen objekt, který bude obsahovat všechny překlady v daném jazyce a bude obsahovat metody pro vypsání hledaného překladu.

JSONTree – Hlavním účelem této třídy bude vytvoření JSON objektu. Tento objekt poté bude sloužit jako zdroj dat pro navigační stromy, které budou vytvořeny pomocí javascriptu.

EditTool – Tato třída bude obsahovat množství metod, jejímž typickým vstupem bude seznam či nějaký jiný datový typ, který bude obsahovat hodnoty. Metody tyto hodnoty vloží do html kódu, který bude jejich výstupem. Účelem třídy EditTool tedy bude zobrazování hodnot v HTML formátu.

Router – třída Router bude obsahovat aplikační logiku pro zpracování žádáné URL. Dále bude kontrolovat existenci žádaných objektů a entit. Z této třídy bude vytvořen objekt, který uchová názvy právě žádaných objektů a entit.

SourceCode – tato třída bude sloužit pro načítání, vytváření a ukládání entit v systému Caché. Výstupy svých metod přepošle metodám třídy EditTool, které je vypíšou v CSP stránce.

ObjectProperty – jedná se o zanořený balíček tříd v balíčku classEditTool. Účelem tohoto balíčku bude práce s názvy vlastností jednotlivých entit.

ObjectProperty.Item – třída bude obsahovat popis vlastnosti entity, jako je například typ entity, jméno vlastnosti atd.

ObjectProperty.Definition – třída bude sloužit ke zpracování vlastností (např. Filtrace dle typu entity, atd.) a hlavně k manuální definici těchto vlastností. Pro každou entitu vyskytující se ve vytvářené aplikace bude vytvořen soubor objektů ObjectProperty.Item.

Při návrhu tříd jsem se snažil oddělit významy a účely jednotlivých operací. Tyto účely jsem poté sjednotil do tříd, kde se každá třída věnuje pouze svému účelu. Instance různých tříd tedy mezi sebou budou komunikovat.

Tento návrh umožňuje jednoduché rozšiřování aplikace o případné další prvky, či úpravy HTML struktury dokumentu.

6.5.2 CSP stránka

CSP stránka bude mít několik funkcí. Bude sloužit k ovládání aplikace a zobrazení výsledků, tedy bude dle potřeby volat metody tříd a jejich výsledkem naplní obsah stránky. CSP stránka bude naplněna kostrou pro strukturu obsahu a vlastní obsah této kostry naplní hodnotami, které jí dodají postupně volané třídy a metody.

6.6 Konfigurace aplikace

Vytvářená aplikace si bude uchovávat svoje nastavení v konfiguraci. Je nutné uchovávat:

- nastavení jazyka
- nastavení výchozího jmenného prostoru po přihlášení
- nastavení rozložení prvků (layout)

Obvyklým způsobem konfigurace aplikací je uložení těchto nastavení do samostatného souboru. Vzhledem k tomu, že aplikace bude na straně serveru a ne u klienta, by bylo nutné uchovávat nastavení pro každého klienta zvlášť a soubor by mohl nekontrolovaně růst. Navíc kdokoliv s povolením zápisu souboru na straně serveru by mohl změnit nastavení i pro ostatní uživatele. Tento přístup jsem tedy zavrhl a zvolil jsem takový typ ukládání, který se bude odehrávat pouze na straně klienta. K tomuto účelu slouží soubory cookies, které také pro tuto aplikaci využiji.

6.7 Překlady

Překlady budou řešeny formou samostatného souboru pro každý jazyk. Jako ukázku použití překladů zpracuji pro aplikaci hlášení ve dvou jazycích – česky a anglicky.

Strukturu souboru s jazyky převezmu z již existujících jazykových překladů pro systém Caché. Soubor bude formátu XML a bude obsahovat dvojice hodnot „heslo“ a „překlad“.

7 Implementace

Celkový návrh a vybrané technologie jsme popsali v předešlé kapitole a nyní si můžeme přiblížit využití vestavěného API systému Caché z implementačního hlediska.

Při vytváření aplikace jsem také narazil na několik zajímavostí, problémů a odlišností od návrhu, jejichž řešení zde popíšu.

7.1 Využití API Caché v aplikaci

Shrňme si nyní návrh práce: webová aplikace bude určitou formou nadstavby nad systémem Caché. Pro operace jako je načítání, ukládání a vytváření entit využívá API Caché, které tyto základní operace přímo podporuje. Jednoduchost použití tohoto API jsme si demonstrovali v kapitole 4.2.1. Jedinou věcí, u které se vyskytly problémy, byly jmenné prostory. Nutnost jejich dynamického přepínání není nikde zdokumentována (je ale možné, že se mi tuto nutnost pouze nepodařilo objevit) a to v kombinaci se znepřístupněním vztahovaných entit po změně jmenného prostoru vedlo k poměrně rozsáhlé změně v aplikační logice. Tato změna spočívala v přepracování metod, které zpracovávají vlastnosti vztahovaných entit (načtení a uložení) – je nutné, aby si tyto metody jmenné prostory přepínaly samy v průběhu vykonávání kódu, jinak by tyto vlastnosti nebyly přístupné. Toto téma je blíže zpracováno v kapitole 4.2.3.

7.2 Přístup k vlastnostem vztahovaných entit pomocí proxy funkcí a metod

Vztahované entity a jejich vlastnosti jsou si v jedné oblasti velice podobné – spojuje je prakticky totožný způsob přístupu za předpokladu, že zobecníme vlastní typy těchto vztahovaných entit a jmen jejich vlastností. Pro zjednodušení přístupu k těmto entitám (a vyhnutí se redundance v kódu) jsem vytvořil *proxy funkce* a *proxy metody*. Těmto funkcím předám libovolný objekt (v tomto případě vztahovanou entitu) a libovolný řetězec, který představuje název vlastnosti entity. Funkce poté může pracovat s hodnotou této vlastnosti předaného objektu. V dalších funkcích používám tento přístup i k dynamickému přístupu pro volání metod, kdy se název volané metody složí z řetězců v průběhu vykonávání odlišné metody.

Tento přístup umožnil oddělit aplikační logiku od aplikačních dat. Využil jsem jej například pro zjednodušení a centralizaci ručního výběru vlastností entit k editaci, či ke zobecnění přístupu ke vztahovaných metodám.

7.3 Získání detailů entity

Načtení detailů vztahované entity je již popsáno v kapitole 4.2.2. Nyní si ale uvedeme plné znění algoritmu v pseudokódu, abychom demonstrovali celý postup pro získání detailů vztahované entity – od načtení třídy až po výpis vlastností v HTML.

```
// JP znamená jmenný_prostor

Function getRelationshipContent (JP_třídy, název_třídy,
název_hledané_entity, typ_hledané_entity)
{
      původní_JP = získej_aktuální_JP()
      přepni_JP(JP_třídy)
      načti_třídu(název_třídy)
      načti_vztahované_entity_dle_typu(typ_hledané_entity) // pomocí
proxy metody
      hledaná_entita = najdi_entitu_dle_názvu(název_hledané_entity)
      // v této chvíli již máme v operační paměti načtenou hledanou
vztahovanou entitu a můžeme s ní dále pracovat
      výstup =
načti_obsah_vlastností_v_html_formatu( typ_hledané_entity) // zavolá se
opět proxy metoda, tentokrát pro zpracování různých vlastností různých
typů entit
      jestli (entita_obsahuje_zdrojový_kód)
      proveď {
            výstup +=
tisk_zdrojového_kódu( hledaná_entita.zdrojový_kód )
      }
      přepní_JP(původní_JP)
      vrať výstup
}
```

7.4 Uložení objektu a výchozí nastavení vlastností

Pomiňme nyní implementační nutnosti při ukládání entity, které jsou obdobné, jako u načítání vlastností entit. Soustřeďme se na samotné přiřazování vlastností do entit.

Vlastnosti entit se totiž mohou nacházet ve dvou obecných stavech, které se až poté dělí na konkrétnější hodnoty dle daného typu vlastnosti. Tyto dva stavy se určují dle přiřazení hodnoty do vlastnosti – přířazená hodnota změní stav vlastnosti na „definovaná“. Ovšem i „nedefinovaná“ vlastnost může obsahovat, a často obsahuje, konkrétní hodnoty. Tyto hodnoty představují výchozí nastavení vlastností, které se přiřadí při vytvoření entity.

Problém nastává v chování aplikace Caché Studio – při výpisu vlastností do zdrojového kódu (uvedeno na obrázku 5) vypisuje pouze ty, které byly definovány. Tedy ty vlastnosti, jejichž hodnoty byly přiřazeny *zvenčí*. Na hodnotě vlastnosti při výpisu nezáleží – je důležitý stav vlastnosti (definovaná / nedefinovaná). Z tohoto důvodu musí aplikace filtrovat vlastnosti, které jsou určené k uložení.

Řešením tohoto problému bylo vytvoření dočasné entity, která sloužila jako zdroj výchozích hodnot. Při ukládání vlastnosti aplikace zkontroluje stav vlastnosti v systému. Jestliže je definovaný, hodnotu přiřadí k vlastnosti. Jestliže je nedefinovaný, tak zkontroluje hodnotu s výchozí a jestli se liší, přiřadí ji do vlastnosti. Jestliže se neliší, nebude provedena žádná změna. Tento algoritmus je popsán v pseudokódu níže s využitím proxy metod a vlastností.

```
Method setProperty (entity e, propertyName pn, value v, defaultValue
dv)
{
            if ( proxyMethod(e, pn+”IsDefined”) || v != dv)
            {
                  proxyProperty(e, pn) = v;
            }
}
```

7.5 Zobrazení a editace zdrojového kódu

Zdrojový kód bude zobrazen v html tagu <textarea>. Pro plnou funkčnost je třeba zajistit vkládání tabelátoru. Výchozí nastavení prohlížečů je, že při stisku této klávesy předají aktivitu dalšímu prvku v DOM stromu HTML dokumentu.Tomuto chování chceme zamezit a místo toho vložit tabelátor na aktuální pozici kurzoru. Vytvořil jsem tedy funkci v javascriptu, která toto chování zajišťuje.

Další problém nastává, když je součástí zdrojového kódu entity nějaký HTML prvek. Je třeba zajistit, aby se tyto prvky zobrazily pouze jako text a neinterpretovaly se – tedy použít escapování znaků. Tuto operaci za použití správných funkcí opět zvládá API Caché.

7.6 Bezpečnost aplikace

Vytvořená aplikace ponechává všechnu správu týkající se autentizace a autorizace na systému Caché. Jestliže uživatel nemá k daným třídám přístup, třídy či metody se mu nezobrazí v navigaci. V případě, že jejich názvy ručně zadá jako součást URL, aplikace vypíše hlášku o neexistující entitě či nedostatku práv pro přístup k ní.

Přidal jsem pouze implementaci bezpečného odhlášení uživatele. Toto odhlášení se provede i po určité době nečinnosti, což je opět způsobeno nastavením systému Caché.

7.7 Nestandardní výpis výsledků kompilace

Původní návrh aplikace počítal s tím, že se všechna aplikační logika (zpracování akce a plnění obsahu) provede ještě před začátkem výpisu HTML struktury dokumentu. Při tomto přístupu jsem ale narazil na metodu pro kompilování tříd, která vždy vypíše svůj výsledek na standardní výstup a musel jsem tedy tento návrh poupravit.

8 Testování

Během vývoje této aplikace jsem využíval 3 fáze testování. V této kapitole všechny tři zmíním, popíšu je i s výsledky a jejich přínosem pro aplikaci. Všechny fáze testování jsou zaměřeny primárně na funkčnost aplikace, vzhledem k zadání práce chápu tento faktor jako stěžejní. Při poslední fázi jsem zkoumal i názor testovacích subjektů na použitelnost uživatelského rozhraní.

Pro první dvě fáze byl testovacím subjektem autor této práce, při poslední fázi jsem požádal o zhodnocení potencionální typické uživatele – programátory.

8.1 1. fáze – testování v průběhu implementace

První fáze testování, probíhající v průběhu vlastní implementace, jistě zabrala nejvíce času. Hlavní důvod této skutečnosti spočívá v mé neznalosti implementačních detailů systému Caché a postupném poznávání vlastností a zákonitostí celého databázového systému. Téma této bakalářské práce se týká přímo jádra systému a při vyhledávání informací jsem často narážel na nezdokumentované části systému. V jednom případě jsem musel požádat i zástupce firmy Intersystems (Ing. Daniela Kutáče) o radu. S přibývajícími znalostmi se ale doba testování a zavádění oprav v této fázi v poměru s dobou strávenou vývojem blížila minimu.

8.1.1 Metoda testování

V této fázi jsem využíval naivní metodu testování – probíhalo pouhé ověření funkčnosti vyvíjené části aplikace. Při dokončení většího úseku či milníku ve vývoji proběhlo testování funkčnosti celé aplikace.

8.1.2 Výsledky testování

Výsledkem tohoto přístupu k testování byla aplikace, která sice splňovala zadání, ale jen za určitých podmínek, které budou upřesněny v další fázi testování.

8.1.3 Přínos

Přínost tohoto testování se pojí s mým postupným hlubším poznáváním chování systému Caché. Pro zajištění funkčnosti jsem musel obměňovat implementační detaily a hledat použití obdobných operací přímo ve zdrojových kódech jádra, což mi umožnilo nahlédnout do programátorského stylu vývojářů, kteří tvořili tento systém, což výrazně zlepšilo mé chápání systému a umožnilo tvorbu složitějších konstrukcí.

8.2 2. fáze – testování uživatelů s odlišnými rolemi

Hlavní osou této fáze je rozšíření použití aplikace na více než jeden jmenný prostor, více než jednoho uživatele a nastavení aplikace, aby vyžadovala uživatele s určitou úrovní oprávnění. Pro nastavení těchto možností jsem využil systémovou webovou aplikaci Management Portal.

8.2.1 Metoda testování

V nově vytvořeném jmenném prostoru jsem pomocí Caché Studia vytvořil testovací třídu, kterou jsem naplnil různými vlastnostmi. Uživatelům jsem nastavil různé role a práva a aplikaci jsem nastavil, aby od uživatelů vyžadovala určitou úroveň oprávnění.

8.2.2 Výsledky

Výsledkem testování bylo zjištění, že aplikace havaruje při přístupu do jiného jmenného prostoru, než je ten, kde se sama nachází (nebo není-li tento jmenný prostor systémový). Toto vedlo k úpravě návrhu tak, aby umožnil přístup i do jiného jmenného prostoru (blíže jsme si tento problém popsali v kapitole 4.2.3).

Naproti tomu aplikace správně pracovala s vyžadováním uživatelských oprávnění.

8.2.3 Přínos

Díky této fázi testování jsem zjistil chybné chování aplikace při přepnutí kontextu do jiného jmenného prostoru. Oprava spočívala v přepracování implementace (popsáno v kapitole 7.1). Po zavedení této poměrně rozsáhlé opravy byla aplikace schopná přepínat kontexty dle libosti uživatele a správně pracovat s uživatelskými účty i s odhlašováním.

8.3 3. fáze – konečné testování

Nyní, když máme ověřenu funkčnost s více uživateli i v rámci různých jmenných prostorů, můžeme se pustit do konečné fáze testování. Tuto fázi jsem testoval na dvou uživatelích, kteří mi po testování sdělili svoje postřehy. Zvolil jsem specifickou metodu testování, která vyplynula z konzultací s vedoucím této bakalářské práce.

8.3.1 Metoda testování

Protože v zadání práce nebyl uveden požadavek na editaci CSP stránek, tuto stránku jsem uživatelům již předpřipravil a v systému jsem vytvořil účty s oprávněním na zobrazení této CSP stránky. S CSP stránkou byla spojena URL adresa, která zobrazovala výsledky jejich snažení. Obsahem CSP stránky bylo vytvoření objektu, který po svém vytvoření zavolal metodu pro výpis textu. Třída, která měla sloužit jako předpis pro tento objekt však dosud neexistovala a právě její vytvoření (a vytvoření její metody) bylo úkolem testovacích uživatelů. Pro zrychlení testu jsem jim poskytl letmý úvod do systému Caché.

8.3.2 Výsledky

Oba uživatelé zdárně dokončili úkol v přiměřeném čase. Neméně důležité byly jejich postřehy při práci v aplikaci. Oba dva se shodli na tom, že aplikaci střídmé uživatelské rozhraní vyhovuje, rychle se v aplikaci zorientovali. Byly ale vzneseny námitky na použité barvy – při editaci vlastností splývá pole pro vložení textu s jeho pozadím. Dále bylo poukázáno na uživatelsky ne zcela přívětivé navigační prvky po stranách aplikace – pravý strom se vztahovanými entitami se rozevře až po rozkliknutí. Přívětivější by prý bylo, kdyby se rozevíral automaticky.

Bohužel, kvůli nedostatku času nestihnu aplikaci upravit.

8.3.3 Přínos

Uživatelé dokázali splnit svůj úkol pouze za pomoci vyvíjené webové aplikace, čímž dokázali funkčnost a použitelnost tohoto nástroje.

Dalším přínosem jsou zkušenosti těchto testovacích uživatelů při práci s aplikací. Zapracuji je do možných rozšíření nástroje.

9 Závěr

V této závěrečné kapitole zhodnotím tuto bakalářskou práci a její výsledky jako celek. Dále si popíšeme známá omezení aplikace a další možnosti jejího vývoje.

9.1 Shrnutí

Cílem této bakalářské práce je tvorba webového vývojářského nástroje pro vytváření a editaci tříd, metod a rutin v systému Caché. Tento nástroj jsem navrhl a implementoval. Z výsledků testování vyplývá, že výsledná aplikace všechny tyto cíle splňuje. Navíc splňuje i rozšířené zadání s možností vytváření a editace všechy typů vztahovaných entit (nejenom metod). Aplikaci jsem vyvíjel tak, aby respektovala a žádným způsobem neovlivňovala uživatelská oprávnění systému Caché, svým zaměřením splňovala požadavky a další možná přání zamýšleného cílového uživatele, instalovala se co nejsnadněšjím způsobem a umožňovala snadné rozšiřování funkčnosti. Systém Caché je rozšířen celosvětově, proto jsem do aplikace zapracoval i možnost překladů uživatelského rozhraní, kdy je v základu dostupná čeština a angličtina.

Vytvořenou aplikaci i s tímto textem poskytnu zástupci společnosti Intersystems, který o ni projevil zájem.

V tomto textu je nejdříve popsán systém Caché, ale pouze ty jeho části, které souvisejí s tématem této práce. Dále proběhl rozbor stávající situace pro možnosti editace entit v Caché. Protože žádný takový webový nástroj dosud neexistoval, proběhla i analýza webových nástrojů s podobným zaměřením. Následuje zpracování potřebného teoretického pozadí k entitám, jelikož manipulace s nimi tvoří zadání této práce. Dále jsem zanalyzoval požadavky a s těmito znalostmi poté navrhl systém. Nastínil jsem téma implementačních detailů a poté jsem zpracoval podrobnější testování aplikace.

9.2 Omezení aplikace

Omezení aplikace se z důvodu splnění zadání soustředí jen na výsledky zkušeností uživatelů při testování. Jak již bylo uvedeno v kapitole 8.3.2., nejvýznamnějším omezením je navigační prvek v pravé části obrazovky, který se neotevírá automaticky, a tak zpomaluje práci.

9.3 Další možnosti vývoje

Možností směrů dalšího vývoje je několik. Uvedu zde několik základních.

Lze pokračovat v dalším vývoji tohoto nástroje – rozšířením aplikace i o vytváření a editaci CSP stránek, aby byl vývojářský potenciál aplikace splněn do posledního možného požadavku. Dále se v tomto směru nabízí zapracování příkazového terminálu do webové aplikace (systém Caché takový prvek umožňuje), dále například zpracování prohledávání v třídách a rutinách.

Dalším směrem, kterým lze další vývoj vést, je výrazné zlepšení uživatelského i vývojářského prostředí této aplikace. Citelné omezení v této oblasti tvoří dva problémy: nepřítomnost javascriptového frameworku pro grafické zpracování syntaxe zdrojového kódu a dále je to nemožnost vrácení změn. Řešením posledního zmíněného je zapracování systému ZenPage do aplikace, což je obdoba AJAXu pro systém Caché. Tento nástroj by výrazně urychlil vývoj aplikací.

Třetím směrem je převzetí možností Caché Studia z pohledu vývojářských funkcí, které tato desktopová aplikace nabízí. Tato aplikace obsahuje množství různých průvodců pro uživatelsky přívětivý vývoj aplikací a tito průvodci by se mohli stát součástí i tohoto webového nástroje. Dále se nabízí možnost zapracovat do aplikace přímé propojení s nápovědou, manuálem a API, což by otevřelo i nové možnosti například pro automatické doplňování textu. Tento směr vývoje ovšem předpokládá splnění předchozích dvou bodů.

Při úspěšném splnění všech třech (nebo alespoň prvních dvou) možných směrů vývoje by se z této aplikace mohl stát úspěšný produkt, který by mohl být součástí instalace systému Caché.

Použitá literatura a zdroje

[1] Kolektiv autorů: The Components of Caché: Caché scripting languages. [online]. [cit. 13.5.2013]. Dostupný z WWW: <http://www.intersystems.com/cache/technology/components/script/index.html> [anglicky]

[2] Wolfgang Kirsten: Caché: databáze postrelačního typu a tvorba aplikací. Brno, CP Books, 2005. ISBN 80-251-0491-5

[3] Kolektiv autorů: Documatic - Online class documentation [online]. 2013 [cit. 13.5.2013]. Dostupný z WWW: <http://docs.intersystems.com/cache20131/csp/documatic/%25CSP.Documatic.cls> [anglicky]

[4] Kolektiv autorů: phpMyAdmin: about [online]. 2013 [cit. 13.5.2013]. Dostupný z WWW: <http://www.phpmyadmin.net/home_page/index.php> [anglicky]

[5] Kolektiv autorů: MooTools - a compact javascript framework [online]. 2013 [cit. 13.5.2013]. Dostupný z WWW: <http://mootools.net> [anglicky]

[6] Kolektiv autorů: "Mif.Tree" – mootools tree control. [online]. 2013 [cit. 13.5.2013]. Dostupný z WWW: <http://mifjs.net/tree/> [anglicky]

[7] Kolektiv autorů: Documatic - JSONObject [online]. 2013 [cit. 12.5.2013]. Dostupný z WWW: <http://www.json.org/javadoc/org/json/JSONObject.html> [anglicky]

V průběhu celé práce a zejména v ranné fázi vývoje aplikace jsem čerpal i ze zdrojových kódů jádra dostupných přes aplikaci Caché Studio.

Seznam příloh

Příloha 1: DVD

Na přiloženém dvd se nalézají následující soubory a adresáře:

- *cache* – instalační soubory systému Caché
- *web-developer-tools.zip* – archiv s potřebnými instalačními soubory vytvořené aplikace
- *technicka-zprava.odt* – zdrojový soubor této technické zprávy
- *technicka-zprava.pdf* – tato technická zpráva
- *INSTALL* – návod k instalaci vytvořené aplikace
- *APPTEST* – návod k nastavení systému Caché pro testování (vytvoření nových jmenných prostorů, uživatelů a vyžadování oprávnění k přístupu)