VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

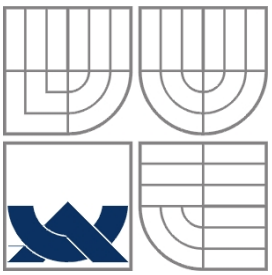
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKAČNÍ PLATFORMA V PHP

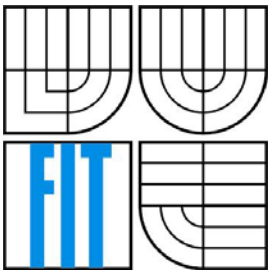
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Pavel Špaček
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKAČNÍ PLATFORMA V PHP

PHP APPLICATION PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Pavel Špaček

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. Zdeněk Martínek

BRNO 2008

Abstrakt

Cílem práce je vytvořit model platformy pro vývoj aplikací za předpokladu, že tato platforma bude realizována v jazyce PHP. Aplikace jsou tvořeny spojováním funkčních modulů dle požadovaných vlastností. Běh takto vytvořené aplikace pak zajišťuje právě platforma.

Klíčová slova

HTML, CSS, PHP, DOM, AJAX, JavaScript, databáze, platforma, webový prohlížeč, administrace, modulárnost, rozšiřitelnost.

Abstract

The goal of this work is to create a model of a platform for an application development provided that this platform is implemented in PHP. Applications are built by connecting of functional modules according to desired properties. The run of these applications is provided by the platform.

Keywords

HTML, CSS, PHP, DOM, AJAX, JavaScript, databases, platform, web browser, administration, modulability, extensibility.

Citace

Špaček Pavel: Aplikační platforma v PHP. Brno, 2008, semestrální projekt, FIT VUT v Brně.

Aplikační platforma v PHP

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Mgr. Zdeňka Martínka.
Další informace mi poskytl Ing. Richard Goldmann.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Pavel Špaček
15. května 2008

Poděkování

Děkuji tímto Ing. Richardu Goldmannovi za pomoc a podporu při vývoji aplikační platformy.

Obsah

1 Úvod

Ve světě informačních systémů je běžnou praxí tvorba aplikací na míru. Tyto aplikace jsou charakteristické tím, že většinou přesně splňují požadavky, které jsou na ně kladené. V oblasti možných rozšíření se opět vychází z vizí zadavatele, které jsou dost často velmi abstraktní a plánované na neurčitou dobu. Tyto vize bývají zahrnuty v návrhu aplikace tak, aby vůbec byly realizovatelné.

Právě největším problémem takových aplikací je rozšiřování funkčnosti a údržba. Vývojáři se tedy snaží o jiné přístupy, které jim pomohou tyto problémy řešit. Výsledkem pak bývají systémy, které se snaží zaměřit na určitý problém a poskytnout nástroje, které zjednoduší a zkrátí čas řešení.

Dobrým příkladem mohou být CMS, tedy Content Management Systémy. Jsou zaměřeny na problematiku tvorby webových stránek a správu jejich obsahu. Tvorba takových stránek je díky těmto systémům zredukována, na přizpůsobení vzhledu podle grafického návrhu. Vytvoření obsahu je pak uskutečněno přes grafické rozhraní, zpravidla vytvořené v JavaScriptu, které používá rozhraní známé u jiných textových editorů. Pokud nějaká funkčnost chybí, je možné ji přidat v podobě zásuvného modulu takového systému. Pokud modul chybí, stačí jej naprogramovat.

Čímž se dostáváme k druhému problému a to náročnosti tvorby těchto systémů jako takových. I tady se snaží vývojáři používat různé nástroje usnadňující práci. Velmi často jsou jimi frameworky, které nabízejí návrhy řešení některých nejběžnějších problémů.

Cílem této práce je pak posunout tvorbu aplikací ještě na vyšší úroveň při použití jazyka PHP. Nabídnout tvůrci sadu funkčních komponent, přirovnatelnou k dílům nějaké stavebnice, a bázi, na které tyto komponenty můžeme poskládat do aplikace, tedy jakousi základní desku stavebnice, na kterou díly skládáme a vzájemně je propojujeme.

V první kapitole se seznámíme s technologiemi, které budou použity pro vývoj tohoto systému. Jedná se o jazyk PHP, HTML, XML, CSS, JavaScript a technologii AJAX.

Další kapitola se věnuje shrnutí požadavků od zadavatele. Požadavky jsou na systém CRM. Právě bližší analýzou těchto požadavků bylo určeno, že je třeba pojmout řešení jinak než obvyklým způsobem. Výsledkem je vývoj PHP aplikační platformy a CRM systému, jako aplikace postavené právě na této platformě. Tato práce se pak zabývá oběma problémy zároveň. Jsou totiž neoddělitelné, neboť CRM chápeme jako test správného fungování platformy.

Následuje kapitola s vysvětlením stěžejních pojmů problematiky frameworků a platforem včetně návrhových vzorů architektury.

V kapitole *Návrh aplikační platformy* jsou vysvětleny struktury a mechanizmy systému implementujícího aplikační platformu.

Návrh architektury platformy zachycuje odvození architektury aplikační platformy od známých architektonických vzorů.

Specifikace požadavků popisuje služby, které systém poskytuje, a omezení, za kterých pracuje. Navazující *Modely PHPAP* poskytují přehled ERD diagramů, ze kterých jsou odvozeny databázové struktury jádra PHP aplikační platformy a zásuvných modulů tvořících CRM aplikaci. Je zde uveden návrh XML souboru, popisujícího zásuvný modul.

V kapitole *Struktura systému* je výčet a podrobný popis objektů, které jsou signifikantní pro jednotlivé části systému. Jde především o vysvětlení logiky implementace PHP aplikační platformy. Popis zahrnuje i vysvětlení metod a upozornění na omezení při tvorbě aplikací nad platformou.

V další kapitole s názvem *Běh aplikace* je podrobně popsáno spuštění zásuvného modulu od inicializace systému, přes spouštění *daemonů* až po zobrazení kýženého výsledku.

Tvorba zásuvného modulu a *Tvorba aplikace* jsou kapitoly navádějící programátora k tomu, dle jakých zásad psát kód zásuvného modulu, co přitom využívat a následně jakým způsobem takto vytvořené moduly spojit ve funkční celek, aplikaci.

PHPAP CRM je kapitolou o jednoduché demonstrační aplikaci na bázi CRM, která ukazuje a potvrzuje funkčnost aplikační platformy jako takové.

V *Závěru* je pak shrnutí práce, dosažených výsledků a vizí do budoucnosti vývoje a používání PHPAP.

2 Použité technologie

Již název a zadání práce naznačují, které technologie budou použity k realizaci řešení tohoto problému.

Skriptovací jazyk **PHP** byl zvolen záměrně. Je to jazyk interpretovaný na straně serveru. Důvodem této volby jsou požadavky na systém. PHP je oblíbený, široce používaný jazyk, vytvořený právě pro účel tvorby webových aplikací. Má poměrně velkou základní knihovnu funkcí. Je zdarma dostupný a nejsou kladena žádná omezení na jeho použití v míře, ve které jej použít potřebujeme. Splňuje požadavek na přenositelnost takto vytvořených aplikací.

Výhodou verze PHP 5.2 a vyšších je možnost nového přístupu k databázím využitím **PDO** a následně frameworku **Doctrine**.

V rámci volby PHP jako programovacího jazyku je pochopitelné, že vstupy a výstupy aplikace budou řešeny za použití webového rozhraní tedy **HTML** případně **XHTML**.

S HTML je asi už neodmyslitelně spjato **CSS**. Zaručí možnost vizuální úpravy dokumentu do přehledné podoby.

Na straně klienta, tedy webového prohlížeče v tomto případě, je vhodné použít **JavaScript**. Jde o skriptovací jazyk interpretovaný přímo webovým prohlížečem. Je s ním úzce spojena technologie **AJAX**. Tato technologie umožňuje zrychlit některé operace, neboť není třeba znovu načítat celé webové stránky, ale je možné získat pouze požadovaná data. JavaScript pak sám o sobě zjednodušuje práci uživateli takového systému rozšířením funkčnosti.

Formát pro komunikaci mezi moduly zaručí **XML**. Je to moderní a přehledný způsob, jak držet strukturu dat a manipulovat s nimi.

Platforma je databázově nezávislá. Tím není myšleno to, že databáze ke svému běhu nepotřebuje, ale to, že není nutné mít instalovaný konkrétní databázový server. Jádro systému je postaveno na databázi **SQLite** a pro vývoj zásuvných modulů platformy jsem zvolil databázový server **MySQL**. Je zdarma dostupný a k danému účelu vyhovující.

2.1 HTML

HyperText Markup Language je značkovací jazyk. Byl stvořený pro prezentaci hypertextových informací na webových stránkách. Byl zamýšlen jako zjednodušení SGML, tedy Standard Generalized Markup Language. Vývoj jazyka je úzce spjat s webovými prohlížeči, které ovlivňují jeho definici.

Dříve se pro tvorbu dokumentů používaly především jazyky TeX, PostScript a SGML. Pro přenos v počítačových sítích však tyto jazyky nebyly vyhovující. Proto přišlo v roce 1990 právě

zjednodušení v podobě HTML s protokolem HTTP, pro přenos takto vytvořených dokumentů. Tvůrce jazyka Tim Berners-Lee tak zužitkoval a rozšířil myšlenku hypertextu Vannevara Bushe.

Jazyk je charakteristický tagy, tedy párovými či nepárovými značkami, do kterých jsou uzavřeny objekty. Tagy jsou parametry těchto objektů a sami jsou dále parametrizovatelné.

HTML 0.9 bylo vydáno v roce 1991. Vytvořena přímo Timem Bernersem-Lee. Tato verze nepodporuje grafický režim.

Průlom jazyka je právě verze **HTML 2.0**, která přináší roku 1994 standardizaci komunitou IETF (Internet Engeneering Task Force). HTML od této verze obsahuje formátování a strukturování dokumentu, možnost práce s obrázky a zpracování formulářů. Neřeší však problém týkající se pozicování objektů. Syntaxe jazyka odpovídá SGML.

Verze **HTML 3.0** přinesla hodně vylepšení v oblasti matematických vzorců, obtékání obrázků, stylů dokumentů a tabulek. Nikdy se ji však nepodařilo implementovat. Proto 14. ledna 1997 přichází verze **HTML 3.2** zachycující stav v roce 1996. Standard je již vydaný konsorciem W3C. Tato verze vypouští většinu vlastností verze 3.0 a spíše standardizuje již funkční implementovaná řešení ve webových prohlížečích. Tyto dvě verze jsou zářným příkladem nezbytnosti konfrontace teoretických a praktických požadavků před standardizací.

HTML 4.0 přináší průlom v definování formátu stránek pomocí kaskádových stylů. Je také možné do dokumentu vkládat externí objekty nebo skripty. Zlepšení se dočkaly především tabulky, formuláře a rámy. Přibyla podpora více jazyků.

HTML 4.1 má za úkol odstranit nedostatky verze předešlé. Původně bylo v plánu ukončit vývoj HTML a pokračovat ve vývoji XHTML. Nakonec však byla vytvořena skupina, která má za úkol vyvinout další verzi oficiálně známou jako **HTML 5.0**.

2.2 XHTML

Extensible HeperText Markup Language spojuje principy HTML a XML. Původně považován za nástupce HTML, nakonec je vyvíjen paralelně a blíží se do verze 2.0. Cílem při návrhu XHTML bylo zachovat kompatibilitu s HTML, ale splnit takové podmínky, aby dokument vyhovoval XML.

2.3 XML

Extensible Markup Language je značkovací jazyk vhodný pro tvorbu vlastních konkrétních značkovacích jazyků. Je navržen především pro výměnu dat mezi aplikacemi. Nezabývá se vzhledem dokumentu, ale především definicí jeho struktury. Nemá tedy definované žádné tagy. Tato volnost dává uživateli jazyka možnost libovolných definic a struktur, přesto však je nutné dodržovat poměrně přísná syntaktická pravidla.

2.4 DOM

Document Object Model, tedy objektový model dokumentu, je objektově orientovaná reprezentace XML, HTML nebo XHTML dokumentu. Jedná se o API, přes které je možné přistupovat k obsahu, struktuře nebo stylu dokumentu a modifikovat je.

Nekompatibilita různých rozhraní v prohlížečích dovedla konsorcium W3C ke standardizaci. Ta vyústila právě ve W3C DOM, přičemž předchozí verze jsou souhrnně pojmenovány jako Intermediate DOM.

DOM v praxi znamená extrakce daného souboru do paměti. Tato technologie se nazývá GROVE (Graphic Representation Of property ValuEs). Alternativní technologií pro zpracování XML a jiných souborů je SAX. Jde o sekvenční průchod souborem, který má menší požadavky na paměť a v určitých případech může být rychlejší. Těmito případy jsou jednoprůchodová čtení souborů. DOM je pak výhodnější pro časté a opakované změny různých hodnot, neboť není třeba celý soubor znovu procházet, ale změní se pouze hodnota na určené adrese paměti.

2.5 CSS

Cascading Style Sheets, tedy tabulky kaskádových stylů, byly vytvořeny v roce 1994 na základě spolupráce Håkona Wium Lie a Berta Bose. Již dříve fungovalo několik jazyků podporujících styly, nikoliv však kaskády. CSS umožňuje řídit vzhled stránek tvůrcům, ale i uživatelům, bez zásahu do zdrojového kóduwebových stránek.

Verze **CSS1** byla dokončena již v roce 1996, ale začala být plně podporována až v roce 1999 v prohlížeči Opera. V roce 2000 byla přijata i prohlížečem Internet Explorer 3.0. Tato první verze se omezovala pouze na úpravu písma a barev. I pozicování se objevilo až později ve verzi CSS-P, která však nebyla nikdy významněji používána. Hlavním problémem se stala různorodost implementací ve webových prohlížečích, vedoucí někdy až k nemožnosti sjednocení vzhledu.

Hlavním znakem **CSS2** je nativní podpora pozicování. Opět se projevuje nesoulad v implementaci v různých webových prohlížečích. Některé vlastnosti byly nakonec vypuštěny a jiné přidány na základě návrhů společností vytvářející webové prohlížeče. Nástupcem se nejspíše stane verze **CSS3**, jejíž vývoj byl odstartován v roce 1998 a stále probíhá.

Aby bylo vůbec reálné zobrazit ve všech prohlížečích, využívá se různých patchů. Například při použití podtržítka před klíčovým slovem, bude v Microsoft Internet Exploreru toto klíčové slovo vyhodnoceno správně, zato v jiných prohlížečích bude považováno za chybu a nebude vyhodnoceno. Tento přístup je ovšem v rozporu s normami, dle kterých je CSS definováno. Čistším postupem je zjištění, pomocí vybraného skriptovacího jazyka, o jaký prohlížeč se jedná a dle toho vybrat příslušný styl dokumentu. Zjištění typu webového prohlížeče je možné provést jak na straně klienta, např.

pomocí JavaScriptu, nebo na straně serveru pomocí analýzy přijaté HTTP hlavičky, ve které jsou zanesené i tyto informace.

2.6 PHP

Personal Home Page (Tools) je interpret stejnojmenného jazyka, který má schopnost dynamicky generovat webové stránky. Spadá do skupiny bezplatného softwaru a je nezávislý na platformě. Jeho zdrojový kód je napsán v modifikaci syntaxe jazyka C. Stejně jako u CGI (Common Gateway Interface) nebo ASP (Active Server Pages) běží i interpret PHP na straně serveru. Nejpoužívanější je v kombinaci s databázovým a webovým serverem pro tvorbu dynamických webových aplikací.

Počátky sahají do roku 1994, kdy Rasmus Lerdorf vytvořil pro svoji potřebu sadu skriptů pro komunikaci s webovým serverem a pojmenoval je **PHP/FI**. Původně byl nosným jazykem Perl, ale pro optimalizaci rychlosti byl celý systém přepsán do jazyka C. V roce 1995 vzniklá verze PHP/FI byla následovaná druhou verzí v roce 1997. Velkým úspěchem bylo rozšíření v té době. Nástroj se stal populárním a byl využit asi na padesáti tisících serverů, což představovalo jedno procento všech domén Internetu.

Zamýšlený projekt jednoho muže se transformoval v roce 1998 na projekt týmový nazvaný nově **PHP3**, který se rozšířil i pod platformu Microsoft Windows. Jazyku přibyla také možnost používání objektově orientované syntaxe.

Cílem **PHP4** byla především optimalizace výkonu a rychlosti ve složitějších aplikacích. Takto upravené jádro běžící na Zend Enginu, jak byl engin pojmenován, spatřilo světlo světa roku 2000. Z rozšíření přibyla podpora HTTP sessions, podpora dalších webových serverů a zvýšení bezpečnosti.

31. 12. 2008 končí oficiální podpora starších verzí PHP. Jediným současným nástupcem je **PHP5**, které přináší hlavně přepracování pohledu na objektově orientované paradigma jazyka. Snahou jazyka je získat vlastnosti, které mají jiné moderní programovací jazyky, mezi něž patří odchytávání výjimek a další. Jádrem je Zend Engine 2.

PHP5 má v dnešní době poměrně rozsáhlou standardní knihovnu funkcí. Je možné připojit i mnoho jiných knihoven pro zpracování textu, grafiky, prací se soubory a databázemi. PHP je oblíbené především díky jednoduchosti použití díky kombinaci vlastností několika jazyků. Nabízí jak objektově orientované tak procedurální paradigma. Závisí tedy jen na rozhodnutí programátora, co zvolí. Bohužel některé vlastnosti objektů, jako například násobná dědičnost, stále nejsou implementovány. [7]

PHP5 také přináší nový pohled na práci s DOM. Bylo vytvořeno API nahrazující knihovny DOM_XML PHP4. DOM API osahuje množství funkcí zaručující výkonnější aplikace pracující s DOM. Zde je vhodné zmínit několik základních objektů a metod. Pro přístup k funkcím DOM API je nutné vytvořit DOM objekt pomocí konstruktoru `DOMDocument`. Kromě tohoto objektu jsou přístupné další objekty jako element DOM, pro vytvoření se používá konstruktor `DOMElement`,

a atribut DOM, za použití konstruktoru `DOMAttr`. Metoda `appendChild` nabízí spojování do stromu pro objekty `DOMDocument` a `DOMElement`. Pro připojení atributu k elementu slouží metoda `setAttributeNode`. Mezi nejpoužívanější metody pro výpis struktury pak patří `saveXML` navracející výsledný dokument. Naopak pro načtení XML dokumentu použijeme metodu objektu `DOMDocument` `load`. [6]

Postup načítání XML dokumentu je demonstrován v následujícím příkladě, vycházejícím z procházení popisového XML dokumentu zásuvného pluginu. V příkladu je procházena stromová struktura XML dokumentu za účelem získání informací o možných nastaveních spouštěných funkcí. Tato data jsou uložena do pole `$this -> m_user_rights_edit`.

```
$dom = new DOMDocument ( '1.0', 'iso-8859-2' );
$dom -> load ( $plg_conf_file );
```

Nejprve je vytvořen nový objekt reprezentující DOM dokument. Musí být správně nastavena verze normy XML a kódování, ve kterém je dokument uložen. Dále je zavolána metoda `load` tohoto objektu, která, po zadání parametru představujícího cestu k souboru typu XML, daný soubor parsuje. Tím vznikne jeho objektově orientovaná reprezentace, která dovoluje manipulaci s uzly, atributy a hodnotami prostřednictvím poskytovaných metod.

```
$exec_function_groups = $dom -> getElementsByTagName (
'exec_function_group' );
foreach ( $exec_function_groups as $exec_function_group )
{
  $functions = $exec_function_group -> getElementsByTagName (
  'function' );
  foreach ( $functions as $function )
  {
```

Následuje procházení stromem dokumentu. Metodou `getElementsByTagName` je získáno pole všech elementů daného jména. Pokud takto získaný element označíme jako nový kořen, můžeme použít výše zmíněnou metodu k dalšímu zanoření.

```
    $function_name = $function -> getAttribute ( 'name' );
    $options = $function -> getElementsByTagName ( 'options' );
    $options = $options -> item(0) -> getElementsByTagName (
    'option' );
```

```
    foreach ( $options as $option )
    {
      $option_pos = $option -> getAttribute( 'pos' );

      $this -> m_user_rights_edit [ $function_name ][ 'option' ][
      $option_pos ][ 'xml_value' ] = $option ->
      getAttribute ( 'value' );

      $this -> m_user_rights_edit [ $function_name ][ 'option' ][
      $option_pos ][ 'description' ] = $option -> nodeValue;
    }
  }
}
```

Výhled do budoucnosti vývoje naznačuje nástupce PHP5 v podobě PHP6, které by mělo přinést pak hlavně větší možnosti zabezpečení, zrychlení enginu PHP a nové vlastnosti především právě v oblasti objektově orientovaném paradigmatu.

2.7 MySQL

Je volně dostupná databáze typu klient-server. Je podporována mnoha programovacími jazyky jako: C, C++, C#, Eiffel, Smalltalk, Java, Lisp, Perl, PHP a další. Je stejně jako PHP multiplatformní. Bývá často právě s PHP kombinována, pro vývoj internetových aplikací.

Databázový systém byl vytvořen švédskou firmou MySQL AB. Původními autory projektu jsou Michael Widenius a David Axmark. Je distribuován pod dvěma licencemi. Distribuce s omezenou výkonností je pod licencí GPL a v plné výkonnosti pod licencí komerční.

Cílem vývoje MySQL byla a je především rychlost zpracování příkazů. Cenou za rychlost jsou pak zjednodušení zálohování a opoždění implementace některých vlastností jako triggery, pohledy a uložené procedury. Ty byly implementovány až v posledních verzích na základě rostoucích požadavků programátorů.

Pro správu databáze MySQL lze doporučit *phpMyAdmin*. Je napsán v jazyce PHP a pro práci s databází využívá webové rozhranní. Tento nástroj je velmi populární a rozšíření v rámci jeho nenáročnosti na instalaci, neboť nevyžaduje žádná speciální rozšíření PHP. Nevýhodou tohoto způsobu správy databáze je především snížená rychlost odezvy na zadané příkazy, které je dána omezenými možnostmi webového rozhranní.

Pro vývoj na lokálním stroji je výhodnější využití jiného způsobu správy databáze. Na platformě Microsoft Windows je možné použít např. *SQLyog*. Jedná se o aplikaci, která nabízí rychlý přístup k databázi. Je zde implementována velká část jazyka SQL v podobě grafického rozhraní. Nabízí mimo jiné i vizuální nástroje pro tvorbu relací mezi tabulkami. [6]

2.8 SQLite

Je volně dostupná relační databáze. Autorem projektu je D. Richard Hipp. Podporuje ji mnoho programovacích jazyků. Je mutliplatformní i díky faktu, že je implementována jako knihovna v jazyce C. Každá databáze je uložena v samostatném souboru. Knihovna implementuje téměř kompletní standard SQL92.

Splňuje *ACID*, tedy dodržuje principy atomičnosti, konzistence, izolovanosti a trvanlivosti transakcí. Jde o známku spolehlivosti databází.

Pro administraci databáze je možné použít správce založeného na webovém rozhranní a psaného v jazyce PHP. Jedná se o aplikaci *phpSQLiteAdmin***.** Nevýhodou je, stejně jako u phpMyAdmin, menší rychlost vyřizování požadavků a příkazů.

Pro lokální správu lze doporučit např. *SQLite Manager*, který je implementován jako rozšíření pro webový prohlížeč Firefox. [9]

2.9 JavaScript

Jedná se o multiplatformní, objektově orientovaný skriptovací jazyk. Autorem je Brendan Eich. Používá se jako interpretovaný jazyk ve webových prohlížečích. Syntaxí patří do rodiny jazyků C/ C++/Java. *JavaScript*, jak je pod tímto jménem známý ve většině prohlížečů, stejně jako *JScript*, což je pojmenování v Microsoft Internet Exploreru, je pouze jednou z implementací standardu *ECMAScript* z roku 1997 asociací ECMA (European Computer Manufacturesrs Association) resp. od roku 1998 ISO, tedy International Standards Organization.

JavaScript je původně obchodní název implementace ECMASriptu společností Netscape. Původně pojmenovaný Mocha, později LiveScript, byl vydán v roce 1995 jako doplněk k jazykům HTML a Java.

ECMAScript je implementovaný i v podobě *Rhinola* založená na Rhino, gcj a Apache. Tato implementace se liší především tím, že umožňují běh na serverové části.

JavaScript obsahuje několik metod pro práci s DOM. Pomocí těchto metod je možné přistupovat k jednotlivým elementům dokumentu a modifikovat jejich vlastnosti nebo vytvářet elementy nové.

Následující metody Javascriptu jsou nejčastěji využívány pro práci s DOM objekty. Pro získání reference na hledaný objekt je možné použít metodu `getElementById()`, která referenci získá na

základě unikátního identifikátoru objektu, nebo `getElementsByTagName()`, využívající naopak jména elementů a navracející pole referencí.

Tvorbu nových textových uzlů umožňuje metoda `createTextNode()`, která vrací referenci na vytvořený prvek. Stejným způsobem je možné vytvořit element metodou `createElement()`, nebo atribut metodou `createAttribute()`.

Jsou dostupné také vlastnosti objektů typu element, které obsahují informace o stromové struktuře DOM. Vlastnost `attributes` je pole se všemi atributy daného elementu, `childNodes` je proměnná, také typu pole, obsahující reference na synovské uzly. S vlastností, které nejsou typu pole, je možné jmenovat např. `id`, pro přístup k identifikátoru elementu, `nodeName`, obsahující jméno elementu, `nodeValue`, držící informace o hodnotě elementu. Pro získání informace o nadřazeném prvku je nutné přistoupit k atributu `parentNode`.

Objekty typu element mají také metody, jejichž příkladem může být `AppendChild()`, pro připojení elementu jako dalšího synovského prvku, `getAttribute()`, která vrací hodnotu atributu předaného jako parametr metody, `removeAttribute()`, jenž odstraní atribut, jehož jméno je uvedeno jako parametr, nebo `removeChild()`, k odstranění prvku identifikovaného referencí.

2.10 AJAX

Asynchronous JavaScript and XML je spojení technologií Javascriptu a XML. Jeho účelem je umožnit změnu obsahu webových stránek bez nutnosti znovunačtení celého obsahu. Prostřednictvím AJAXu celá operace proběhne na pozadí. Server tak zašle zpět data, která byla změněna.

Snahou technologie je zvýšit plynulost aplikace, kde se právě díky načítaní na pozadí webové aplikace začínají blížit desktopovým. Tímto způsobem je také možné snížit zátěž webového serveru resp. celé sítě. Ovšem při nevhodné implementaci toto může mít efekt opačný.

Za nevýhody je považována změna paradigmatu používání webu, které se liší od konceptu posloupnosti stránek, jenž je možné procházet tlačítky **Další** a **Zpět**. Stejně jako u náročných operací u desktopových aplikací je nutné uživatele informovat o stavu vyřízení jeho požadavku, tentokráte ale z důvodu síťové latence, zaviněné používáním internetu. AJAX je podporován pouze novými moderními prohlížeči, na což je při tvorbě stránek nutné myslet.

Myšlenka podobná AJAXu je známá již od dob Microsoft Internet Exploreru 3.0 z roku 1996. Podoba AJAXu však jako taková vznikla až v roce 2005 v článku Jesseho Jamese Garretta.

Princip AJAXu je založen na objektu `XMLHttpRequest`, který obstarává komunikaci mezi serverem a klientem. Žel zde ještě neproběhla standardizace, proto se vytvoření instance různí prohlížeč od prohlížeče. Internet Explorer jej vytváří jako objekt Active X, zatím co ostatní prohlížeče jej instancují jako objekt JavaScriptu. [8]

V následujícím příkladu je demonstrováno řešení odeslání formuláře pomocí technologie AJAX a metody POST.

```
<script type="text/javascript">
function ajax ( stranka, kam )
{
  var httpRequest;
  var postEl = document.getElementById ( "sel_type" );
  postData = postEl.name + "=" + postEl.value;
```

Funkce `ajax`, je hlavní části celého procesu. Parametr `stranka` udává, který skript a na jaké adrese bude volán. Parametr `kam` umožňuje volbu místa zobrazení získaných dat. Proměnná `postEl` je referencí na objekt ve struktuře DOM, jehož jméno a hodnota je získána a transformována na řetězec, uložený do proměnné `postData`, který je poslán jako proměnný metodou POST.

```
  if ( typeof window.ActiveXObject != "undefined" )
  {
    httpRequest = new ActiveXObject ( "Microsoft.XMLHTTP" );
  }
  else
  {
    httpRequest = new XMLHttpRequest ();
  }

  httpRequest.open ( "POST", stranka, true );
  httpRequest.setRequestHeader ( "Content-Type", "application/x-www-
  form-urlencoded");

  httpRequest.onreadystatechange = function ()
  {
    processRequest ( httpRequest, kam )
  };
  httpRequest.send ( postData );
}
```

V další části kódu funkce `ajax` je vytvořen objekt `ActiveXObject` resp. `XMLHttpRequest`, což záleží na tom, zda skript běží pod Microsoft Internet Explorerem nebo

jinými webovými prohlížeči. Dále je vytvořeno spojení se serverem, ve kterém je definována metoda přenosu dat `open()`.

Tato metoda, pro správnou inicializaci spojení, potřebuje znát typ metody přenosu, a adresu, na kterou je spojení navázáno. Volitelně je možné zadat hodnotu typu `boolean` na pozici třetího parametru, která rozhoduje o synchronnosti přenosu, a další dva parametry reprezentující uživatelské jméno a heslo.

Pokud je přenos nastaven jako asynchronní, nečeká se na odezvu serveru a pokračuje se ve zpracování požadavku. Smysl nastavení přenosu na synchronní je např. při potřebě ověření uživatelského vstupu před pokračováním.

Pro potřebu odeslání dat metodou POST je třeba vytvořit hlavičku pomocí metody `setRequestHeader()`. Následně je zavolána funkce `processRequest()`, ve které zpracujeme data, která byla přijata.

```
function processRequest ( httpRequest, kam )
{
  if ( httpRequest.readyState == 4 )
  {
    if (( httpRequest.status >= 200 && httpRequest.status < 300 ) ||
          httpRequest.status == 304)
    {
      document.getElementById ( kam ).innerHTML =
      httpRequest.responseText;
    }
    else
    {
      alert ( "Chyba pri nacitani stanky " + httpRequest.status +" :
              "+ httpRequest.statusText);
    }
  }
```

Pokud je akce hotova, tedy `httpRequest.readyState` je rovno 4, a server nevrátil žádnou chybu, je výsledek vypsán do elementu, jehož id odpovídá obsahu proměnné kam. V případě chyby, je tato skutečnost vypsána do téhož elementu. V době vyřizování požadavku je v témže elementu vypsána žádost o počkání.

```
  else
  {
```

```
    if ( kam != "" )
    {
       document.getElementById( kam ).innerHTML = "please wait";
    }
  }
}
</script>
```

Ve výše uvedeném příkladu nejsou zmíněny některé metody a vlastnosti, které jsou pro objekt `XMLHttpRequest` taktéž důležité. Jedná se o metody `abort()`, přerušující aktuální požadavek, `getAllResponseHeaders()`, která vrací hlavičky v podobě párů klíč/hodnota, a `getResponseHeader()`, vrací konkrétní hlavičku definovanou identifikátorem.

Dále pak vlastnosti `rensponseText`, nesoucí odpověď serveru ve formě textu, `responseXML`, odpověď ve formě XML, `status`, vracející stavový kód zaslaný serverem, a `statusText`, což představuje textovou verzi stavového kódu.

2.11 PDO

Zvláštní kapitolu si zasluhuje téma PHP Data Objects (PDO). Jedná se o odlehčené a konzistentní rozhraní pro přístup k databázím implementované jako knihovna pro PHP, napsána v jazyce C. Každý ovladač databáze implementuje toto rozhraní. Vlastnosti databáze, které nejsou ve společném rozhraní, pak představují rozšiřující metody daného ovladače.

PDO poskytuje abstrakci pro přístup k datům, tedy nezáleží na používané databázi a vždy je použita totožná funkce k získání dat. PDO na druhou stranu neposkytuje abstrakci databáze, tedy neemuluje u jednotlivých databází chybějící funkčnost oproti databázím jiným.

Toto rozšíření je dostupné od verze PHP 5.0 v knihovně PECL. PHP Extension Community Library je skladiště PHP rozšíření, které nabízí adresář všech známých rozšíření a příslušenství pro jejich vývoj. Ve verzi PHP 5.1 je již zahrnuto v přímo v distribuci.

PDO vyžaduje nově implementované objektově orientované vlastnosti jádra PHP 5.0, proto jej nelze provozovat na starších verzích PHP.

I když v současné době ještě nedosahuje funkčnosti některých rozsáhlých knihoven pro přístup k databázím, jakými jsou např. *mysql.dll* nebo *mydqli.dll*, předpokládá se, že v budoucnu bude právě PDO hlavním, možná i jediným, způsobem jak definovat a manipulovat data v databázích. Např. nová databáze SQLite nemá jinou implementovanou jinou knihovnu pro PHP než *php_pdo_sqlite.dll*.

```
<?php
try
  {
  $dbh = new PDO( 'odbc:SAMPLE', 'db2inst1', 'ibmdb2',
         array( PDO::ATTR_PERSISTENT => true ));
  echo "Connected\n";
  $dbh -> setAttribute ( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

V první části příkladu je vytvořen nový objekt PDO a s ním perzistentní připojení k databázi. Následně jsou nastaveny atributy databáze pro odchytávání chyb.

```
  $dbh -> beginTransaction ();
  $dbh -> exec ( "insert into staff ( id, first, last ) values ( 23,
                 'Joe', 'Bloggs' ) ");
  $dbh -> exec( "insert into salarychange ( id, amount, changedate )
                 values ( 23, 50000, NOW() ) ");
  $dbh -> commit ();
```

V další části je vytvořena transakce složená ze dvou příkazů insert. V téže části je volán příkaz pro vykonání transakce. Pokud vykonání selže, je transakce odstraněna metodou `rollBack()` a systém je uveden do předcházejícího stavu, čímž je zachována konzistence dat.

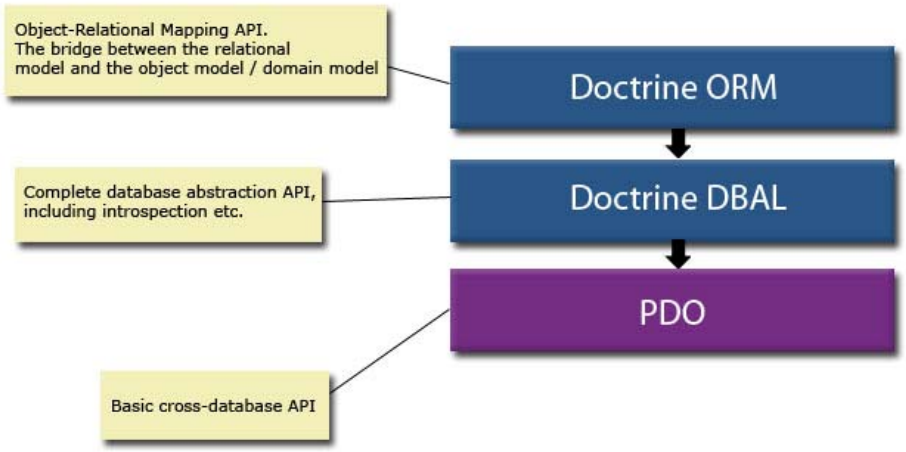
```
}
catch ( Exception $e )
{
  $dbh -> rollBack ();
  echo "Failed: " . $e -> getMessage ();
}
?>
```

2.12 Doctrine

Doctrine, nebo také známo jako phpDoctrine, je ORM nad databázovou abstrakcí a framework pro PHP 5.2.3 a vyšší implementovaný v jazyce PHP. K přístupu k databázím využívá PDO.

Object-relation mapping (ORM) je významem programovací technika, která vzájemně konvertuje nekompatibilní objektově orientované a relační datové typy. V důsledku je vytvořena virtuální objektová databáze nad databází relační, které je přístupná skrze syntax daného objektově

orientovaného jazyka. Objekt pak odpovídá databázové tabulce. Logika tohoto přístupu k databázím je známa jako návrhový vzor *Active record pattern*. [10]



Obrázek 2.1 Struktůra frameworku Doctrine

Doctrine bylo vytvořeno v dubnu roku 2006. Doplňuje PHP o prostředky, které se dříve uplatnily v jazyce Java. V současné době je ve verzi 0.11.

Doctrine není omezením, ale rozšířením. V případě potřeby je vždy možné využít přímo PDO nebo normovaného SQL, které je dostupné přes metodu `Doctrine_RawSql()`. Nabízí vlastní objektově orientovaný způsob zapisování SQL příkazů nazvaný Doctrine Query Language (DQL).

2.12.1 Příklad použití

V následujícím příkladu použití je vytvořeno spojení a provedeny příkazy `select` a `insert`.

```
$manager = Doctrine_Manager :: getInstance ();
```

Vytvoříme instanci manažera databázových připojení. Ten umožňuje zjednodušenou správu více připojení k databázím.

```
Doctrine_Manager :: connection ( $dns , 'jméno_spojení' );
$dsn = 'mysql :// username : password @ localhost / test ';
```

Zavoláme metodu pro vytvoření spojení. Prvním parametrem je řetězec definující spojení, druhý je pak unikátní pojmenování spojení. Dále je nutné připojit soubory s třídami, které popisují tabulky relační databáze.

```
Doctrine :: generateModelsFromDb ( 'cesta/kam/vygenerovat/soubory '
);
```

Třídy popisující databázové tabulky je možné vytvořit ručně, nebo použít výše uvedenou metodu, která je automaticky vygeneruje z databáze, ke které je aktivní připojení.

```
$manager -> setCurrentConnection ( 'jméno_spojení' );
$connection = $manager -> getCurrentConnection ();
```

Dalším krokem je nastavení aktivního spojení a získání reference na objekt aktivního spojení, se kterým se bude dále pracovat.

```
$connection -> query ('SELECT * FROM  test);

$db_object = new $this -> test();
$db_object -> test_name  = 'testovací_jméno';
$db_object -> test_value = 'testovací_hodnota';
$db_object -> save();
```

Završením ukázky je nejprve provedení dotazu typu `select`, při použití normovaného jazyka SQL nad tabulkou *test*. Druhý příklad je příkaz typu `insert`, pomocí kterého jsou hodnoty vloženy do téže tabulky.

2.12.2 Příklad ORM

Níže je uveden příklad tabulky test, použité ve výše demonstrovaném ukázkovém kódu. Třída popisuje tabulku o třech sloupcích. Prvním unikátním identifikátoru záznamu typu `integer`, druhém s názvem `test_name` typu `varchar` délky 255 znaků a třetím opět typu `varchar` stejné délky ale jména `test_value`.

```
abstract class Test extends Doctrine_Record
{
  public function setTableDefinition()
  {
    $this -> setTableName ( 'test' );
    $this -> hasColumn ( 'id', 'integer', 4, array ( 'alltypes' =>
             array (  0 => 'integer', ), 'ntype' => 'int(11)',
             'primary' => true, 'autoincrement' => true ));
```

```
    $this -> hasColumn ( 'test_name', 'string', 255, array (
             'alltypes' =>  array (  0 => 'string', ), 'ntype' =>
             'varchar(255)'));
    $this -> hasColumn ( 'test_value', 'string', 255, array (
             'alltypes' =>  array (  0 => 'string', ), 'ntype' =>
             'varchar(255)' ));
  }
}
```

3 Analýza požadavků

Motivací celého projektu aplikační platformy je analýza požadavků konkrétního zadání a hledání řešení tohoto problému.

3.1 CRM

Customer Relationship Management, neboli systém pro správu zákazníků je původním problém k řešení. Tento systém má být realizovaný v podobě tzv. Troube Ticket Systému, známého především jako informační systém pro helpdesk.

Systém musí být schopný sledovat požadavky zákazníků. Tyto požadavky jsou vyhodnocovány z hlediska stupně provedení. V závislosti na urgentnosti požadavku pak systém podniká kroky, které vedou k informování za požadavek zodpovědných lidí tak, aby byly vyřízeny včas, nebo řešeny jiným způsobem.

Na tuto základní schopnost systému se váží další moduly, které jeho funkčnost rozšiřují. Jde o modul pro zaznamenávání informací o zboží, které je v oběhu, tedy je pronajato či zapůjčeno, modul pro sledování stavu oprav a reklamací, modul správy dat o zákaznících a fakturační modul.

Přístup do systému musí být realizován přes webové rozhraní. Je odlišen interní přístup, tedy požívání systému zaměstnanci firmy, a externí přístup, který je aplikací různých omezení přizpůsoben pro zákazníky.

Bezpečnost zaručuje nepovinné řízení přístupu, založené na právech přidělovaných skupinám uživatelů. Posílení bezpečnosti zajišťuje zaznamenávání všech signifikantních provedených aktivit.

Další skupinou požadavků je provázání s jinými aplikacemi. Nejprve je žádoucí mít systém spojený s ekonomickým softwarem. Řešením je navržení protokolu ke komunikaci, který bude založený na jazyce XML. S problematikou propojení dvou aplikací souvisí problém synchronizace dat.

V rámci poptávky a ekonomického pohledu na věc je zásadním požadavkem možnost upravení systému a jeho použití v jiných firmách.

Přístupem k tvorbě softwaru musí být agilní vývoj. Důraz je především kladen na spolupráci zadavatele a programátora a reagování na změny.

3.2 Jednoúčelové systémy

Jednoúčelový systém je řešení, které bývá obvyklé pro podobná zadání. Jedná se o navržení architektury systému tak, aby splňovala přesně definované požadavky. Rigidnost této architektury je pak kompenzována modularitou systému, případně možností připojení různých rozšíření.

Řešení je výhodné z hlediska optimalizací. Systém má přesně daný účel, na jehož základě je možné přizpůsobit konceptuální model dat v databázi. Zdrojový kód může být psán tak, aby co nejefektivněji plnil požadované funkce.

Nevýhodou takového přístupu je pak především složitost udržování a refaktorizace. Změny v takovém systému mohou totiž znamenat i porušení původní architektury.

3.3 Univerzální systémy

Univerzalita je abstraktní pojem, u kterého je nutné stanovit přinejmenším její stupeň. Musíme brát v úvahu fakt, že absolutně univerzální software neexistuje. Dostáváme se k paradoxu, že i univerzální systém je jednoúčelový v závislosti na míře abstrakce.

V tomto případě jde o univerzalitu ve smyslu, že není předem určeno, jakou funkci bude software plnit. Účelovým omezením je pak jazyk PHP, fixace na databáze a webové rozhraní. Proto je možné říci, že se nejedná o jednoúčelový software, ale platformu, která poskytuje možnost účelovost vytvořit.

Pro lepší pochopení je vhodné vzít za příklad CMS, tedy Content Management System. Tento software je univerzální ve smyslu svého použití, neboť nespecifikuje, jaké stránky zobrazuje. Pokud mu chybí určitá funkcionalita, může být dodána v podobě zásuvného modulu, komponenty apod. CMS je ale na druhou stranu zaměřen pouze na správu obsahu webových stránek.

4 Aplikační platformy

Výsledkem analýzy návrhu, je volba univerzálního systému v podobě aplikační platformy. Abstrakce zůstává na úrovni použitých technologií, tedy neudává účel tvořené aplikace.

4.1 Aplikační platforma

Obecně se softwarovou platformou rozumí prostředí pro běh aplikací poskytující prostředky pro vývoj.

Základ PHP aplikační platformy pak chápejme jako prostředí, které je tvořeno na jedné straně databázovým serverem a webovým serverem s instalovaným interpretem jazyka PHP a na straně druhé webovým prohlížečem.

PHP aplikační platforma je pak tvořena výše zmíněným základem a prostředím vytvořeným na tomto základě v jazyce PHP.

4.2 Framework

Je prostředkem pro běh aplikací. Je popsán platformou. Je také definován jako softwarová struktura sloužící jako podpora při programování a vývoji aplikací. Často obsahuje podpůrné programy, knihovny, návrhové vzory nebo grafické vývojové prostředky.

Pokud se budeme zabývat frameworky pro PHP, znamená to, že budeme mluvit především o knihovnách, které mohou a nemusí předepisovat způsob jejich užívání.

Nejednoduššími frameworky jsou knihovny návrhových vzorů. Mají za úkol vytvořit jistou úroveň abstrakce ve formě obecného řešení, tak aby programátor nemusel řešit často se vyskytující problémy od začátku. Vyhledá příslušnou kolekci návrhových vzorů vztahující se k danému problému, která je reprezentována většinou knihovnou, a vybere si příslušnou návrhový vzor. Tyto knihovny jsou převážně založeny na objektově orientovaném paradigmatu ve formě abstraktních tříd.

Složitější frameworky pak přidávají ke knihovnám funkcí i architekturu, podle které se používají. Jednou z nejznámějších je **MVC**, tedy Model-View-Controller.

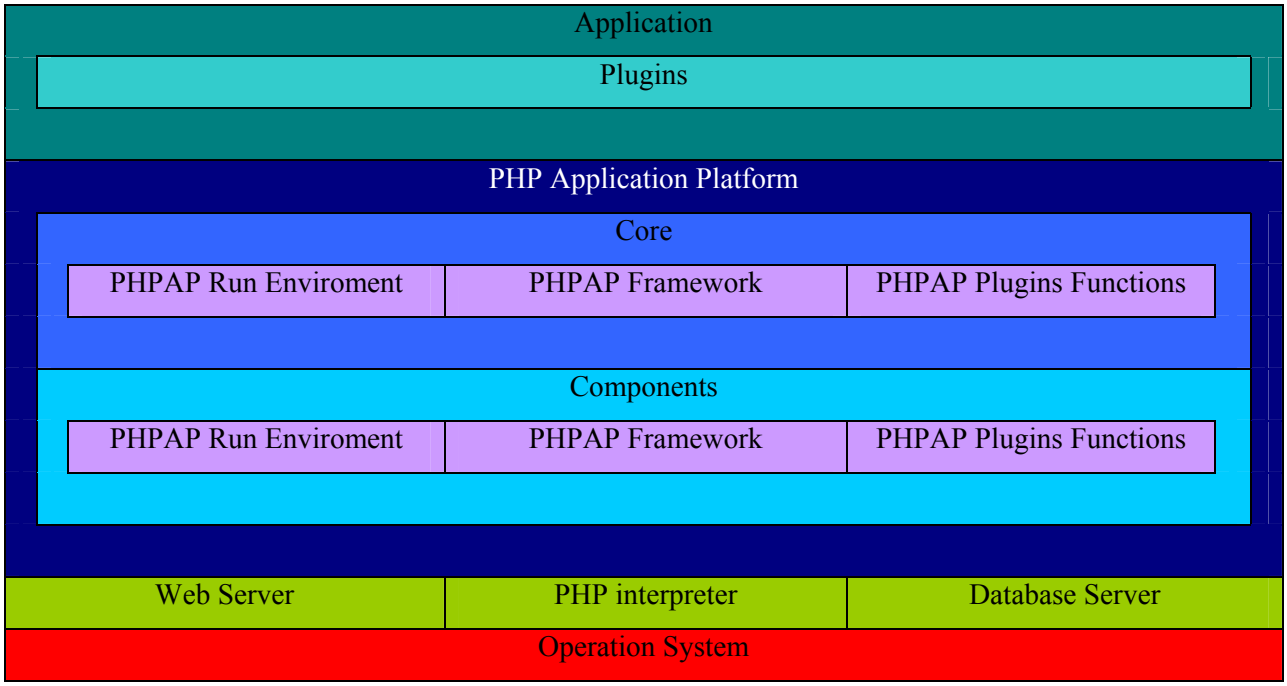
I pro PHP se objevila řešení v podobě frameworku zahrnující také grafické vývojové prostředí. Tyto frameworky jsou však stále ve vývoji a jejich praktické použití je zatím spíše problematické. [2]

5 Návrh aplikační platformy

Základ aplikační platformy je popsán v předchozí kapitole. Cílem je tedy vytvořit její vrstvu na úrovni aplikace zapsané v jazyce PHP.

5.1 Schéma aplikace PHPAP

Na obrázku 5.1 je znázorněno schéma aplikace běžící na PHPAP (PHP aplikační platformě).



Obrázek 5.1 Schéma aplikace PHPAP

PHPAP je složena z jádra, *Core*, a komponent, *Components*. Jádro je složeno z enginu, *PHPAP Run Enviroment*, frameworku, *PHPAP Framework*, a kolekcí funkcí pro práci se zásuvnými moduly, *PHPAP Plugins Functions*. Jádro pak tvoří nepostradatelnou část, která je vždy využita aplikací. Funkce jádra však mohou být rozšířeny pomocí komponent PHPAP. Mají stejnou strukturu jako jádro, ale nemusí být nutně využity pro běh aplikace.

Engin je algoritmus, který na základě vnějšího podmětu rozhoduje, která akce bude provedena, a které funkce budou zavolány.

Framework je tvořen kolekcí návrhových vzorů pro práci a nižší vrstvou, která se skládá z webového serveru, PHP interpretu a databázového serveru.

Kolekce funkcí pro práci se zásuvnými moduly je pak speciálním druhem frameworku pro práci s vyšší vrstvou tvořenou zásuvnými moduly spojených do aplikací.

Funkcí PHPAP je tedy poskytnout prostředí pro běh aplikace, *Application*, a uživatelské rozhraní pro sestavení aplikace ze zásuvných modulů, *Plugins*.

Aplikace, vytvořená nad PHPAP, je složena ze zásuvných modulů. Ty jsou konfigurovány a vzájemně funkčně svázány a to jak mezi sebou, tak s jádrem. Vytváření aplikací není tedy nic víc než spojování a konfigurování zásuvných modulů podle požadavků, které jsou na výslednou aplikaci kladeny.

Každý **zásuvný modul** plní určitou funkci, pro kterou byl naprogramován. Tato jeho funkčnost je nastavitelná, dle parametrů, které nabízí. Zásuvný modul tedy tvoří znovupoužitelnou funkční jednotku, psanou stejně jako PHPAP v jazyce PHP.

PHPAP také definuje architekturu modulu. Ta musí být dodržena, aby zásuvný modul mohl být vůbec spuštěn.

6 Návrh architektury platformy

6.1 Vzory softwarové architektury

Vzor softwarové architektury vyjadřuje základní strukturálně organizační schéma pro softwarový systém. Tento systém je složen z podsystémů, jejich odpovědností a jejich vzájemných vztahů. Nejedná se o architekturu jako takovou. Je to pouze její koncept, který zachycuje její zásadní elementy. Tedy různé architektury mohou implementovat stejný vzor.

Hlavním aspektem vzorů softwarové architektury je to, že ztělesňují atribut kvality řešení. Je důležité ve fázi návrhu správně zvolit tento vzor na základě požadovaných vlastností.

Nejznámějšími takovýmito vzory jsou: Model-View-Controller, Three-tier, Presentation-abstraction-control, Pipeline, Peer-to-Peer a jiné. [5]

6.1.1 MVC

Model-View-Controler je jeden z nejpoužívanějších vzorů pro webové aplikace vůbec (viz obrázek 6.1). Je znám také jako *Model-2*. Odděluje datový model aplikace, uživatelské rozhraní a řídící logiku tak, aby modifikace jedné složky měla minimální řídící vliv na složku druhou. Poprvé byl popsán roku 1979 Trygve Reenskaugem pracujícím v jazyce Smalltalk pro firmu Xerox.

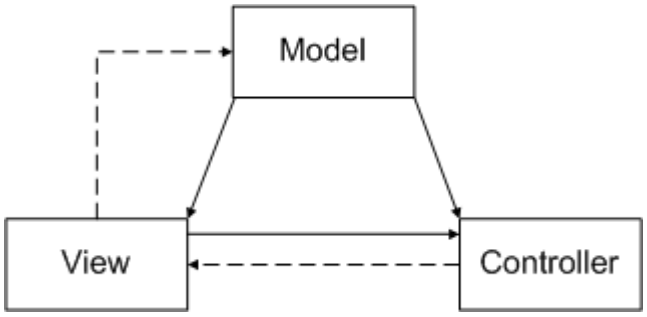
Model představuje doménově specifickou reprezentaci informací, což u webových aplikací znamená především získání požadovaných dat z databáze a jejich organizace do struktur, které se předají k zobrazení.

View, tedy pohled, má na vstupu data zpracovaná modelem, a převádí je do formy, která je prezentována na výstupu. V praxi tedy konečný vzhled webové stránky.

Controller, tedy řadič, očekává příchozí požadavky, typicky od uživatele, a na jejich základě modifikuje model a volá příslušný objekt pohledu.

Obecně je model nezávislý na pohledu, tedy pohled posílá požadavky modelu a on na ně reaguje. Je možné přidat i zpětnou komunikaci, kdy model informuje pohled o případných změnách, které nastaly.

Řadič přijímá vnější požadavek a rozhoduje o akci, která se má provést. Vyšle zprávu modelu. Model na základě této zprávy připraví data, která jsou požadována pro daný typ zprávy. Řadič zasílá zároveň zprávu i pohledu, který sestaví šablonu zobrazující se na výstupu. Jakmile je šablona připravena, pohled posílá zprávu modelu. Na základě této zprávy jsou získána data, která doplňují šablonu. Výsledek je zobrazen na výstup. [1]

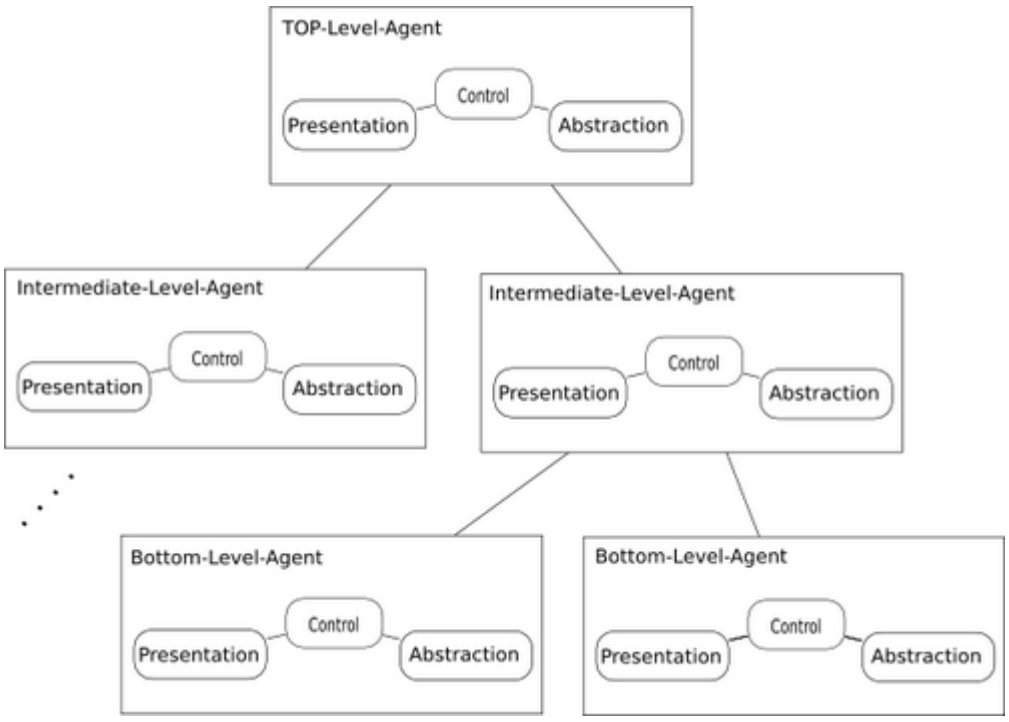


Obrázek 6.1 Model-View-Controller

6.1.2 PAC

Presentation-abstraction-control je architektonický vzor Joëlla Coutaze z roku 1987 (viz obrázek 6.2). Je podobný MVC. Je složen z hierarchie agentů. Každý agent je vnitřně složen z řízení, **Control** odpovídající Controller v MVC, vzhled, **Presentation** odpovídající View v MVC, a abstrakce, **Abstraction** odpovídající Model v MVC.

Agenti spolu komunikují pomocí řízení. Narozdíl od MVC není žádná komunikace mezi abstrakcí a vzhledem. [4]

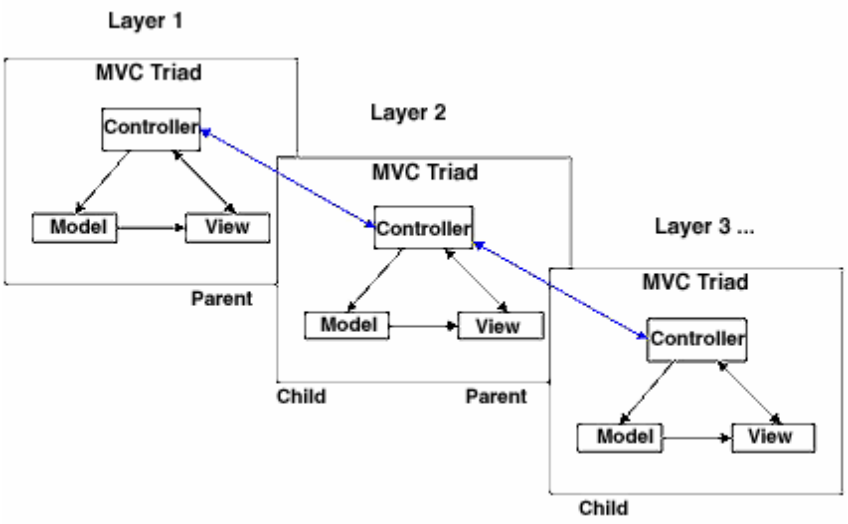


Obrázek 6.2 Schéma PAC

6.1.3 HMVC

Hierarchical-Model-View-Controler je z roku 2000. Zachovává princip MVC a přidává hierarchii mezi vrstvami založenou na vztahu rodič-potomek (viz obrázek 6.3).

Patří do podmnožiny PAC architektonických vzorů. Není tak striktní a dovoluje komunikaci pohledu a modelu mezi sebou, přičemž obchází řadič, tedy zachovává na této úrovni princip MVC. [3]



Obrázek 6.3 Schéma HMVC

6.1.4 Architektura PHPAP

Jako základ architektury PHPAP jsem zvolil modifikaci a kombinaci dvou návrhových vzorů, a to PAC a HMVC. Touto kombinací vzniká návrhový vzor, který je hierarchický. Hierarchie je dána jeho vrstevnatostí. Každá vrstva může mít více trojic MVC.

První vrstvou je pak řadič jádra. Ten přebírá URL adresu, ze které získá proměnné. V kombinaci s hodnotami superglobálních proměnných má všechny informace potřebné k rozhodování o akci.

Pokud má jádro něco vykreslit, pak tak učiní. Toto je především případ běhu jádra v konfiguračním módu, který umožňuje nastavení vlastností zásuvných modulů a jejich vzájemné svazování. Pak přichází zavolání řadičů zásuvných modulů.

Každý zásuvný modul je reprezentován jednotkou postavenou na architektonickém vzoru MVC. Jeho prostor je vyhrazený řadičem jádra. Jádro pak může volat několik zásuvných modulů najednou.

V praxi tato vlastnost dovoluje vytvořit sestavu, složenou z různých tabulek dat. Každá taková tabulka dat je přitom reprezentována výstupem jiného zásuvného modulu zavolaného ve speciálním módu.

Při výběru architektury jsem vycházel ze zkušenosti svojí a tvůrců frameworků a různých informačních systémů, která říká, že MVC, případně její modifikace, je nejspíše nejvhodnější právě pro tvorbu aplikací tohoto typu.

Prvotní úvahou bylo použít pouze architektonický vzor MVC. Celý systém by tedy byl řízený jedním řadičem a zásuvné moduly by byly pouze další svazky pohledů a modelů. Výhodou tohoto přístupu je právě vynechání vrstevnatosti aplikace. Nemusela by se tedy řešit komunikace mezi řadiči. Problémem tohoto přístupu je zvýšení náročnosti modifikací jádra systému. Za předpokladu, že i jádro bude muset být vyvíjeno, je žádoucí jeho oddělení jako samostatné zapouzdřené části.

7 Specifikace požadavků

PHPAP musí být spustitelná na webovém serveru *Apache* verze 2.0 nebo vyšší, na kterém je nainstalovaný *PHP interpret* verze 5.2 a vyšší. Musí být schopný spolupracovat s databází *MySQL* a *PostgeSQL*. V další fázi by měl spolupracovat i s databázemi *MSSQL*, *Oracle* a dalšími, které jsou PHP interpretem podporovány.

Požadavky kladené na webový server a interpret PHP musejí být minimální, ve smyslu nestandardních nastavení. Vychází se z předpokladu, že aplikace může být instalována na různých webových serverech s různými nastaveními, proto musí její jádro využívat pouze základní standardní knihovny funkcí. Myslí se tím knihovny nainstalované při běžné standardní instalaci.

Jakékoliv používání nestandardních knihoven je možné pouze pro zásuvné moduly. Tato skutečnost musí být uvedena v popisu modulu a modul sám musí hlídat, zda je taková knihovna nainstalována. V případě že není, nesmí dovolit pokračování v instalaci nebo musí znemožnit používání svých funkcí, které jsou na této knihovně závislé.

Omezení vztahující se na používání databází vychází z požadavku jejich variability. Je nutné používat pouze takové funkce, které jsou všem databázím společné. Bude se tedy vycházet z faktu, že všechny databáze podporují standard SQL. Při tvorbě systému budeme využívat pouze možností tohoto jazyka, nikoliv však různých jeho rozšíření jako PL-SQL a jiných.

Systém musí mít danou pevnou adresářovou strukturu, která bude reflektovat funkčnost systému. Musí oddělovat třídy enginu jádra od frameworku a funkcí pro práci se zásuvnými moduly. Sami zásuvné moduly pak musí mít předem definované úložiště a adresářovou strukturu.

U zásuvných modulů se vychází z předpokladu, že se jedná o samostatné funkční jednotky, které dodržují adresářovou strukturu a architekturu MVC. Každý zásuvný modul musí mít ve svém kořenovém adresáři soubor index.php, který je hlavním souborem řadiče a řídí funkčnost zásuvného modulu. Dále se v tomto adresáři bude nacházet soubor, ve kterém je v XML formátu uložen popis zásuvného modulu.

Systém musí mít instalátor, který ve formě návodu, později pak samostatného programu, provede uživatele instalací PHPAP. Instalátor, realizovaný aplikací psanou v jazyce PHP, nesmí být svázaný s PHPAP. Jde o samostatnou aplikaci, která po splnění své funkce bude z bezpečnostních důvodů smazána.

Knihovny funkcí musí v základu obsahovat třídy, které umožní uživateli instalaci, nastavení a provázání jednotlivých zásuvných modulů. Tyto funkce budou spouštěny jádrem se speciální direktivou.

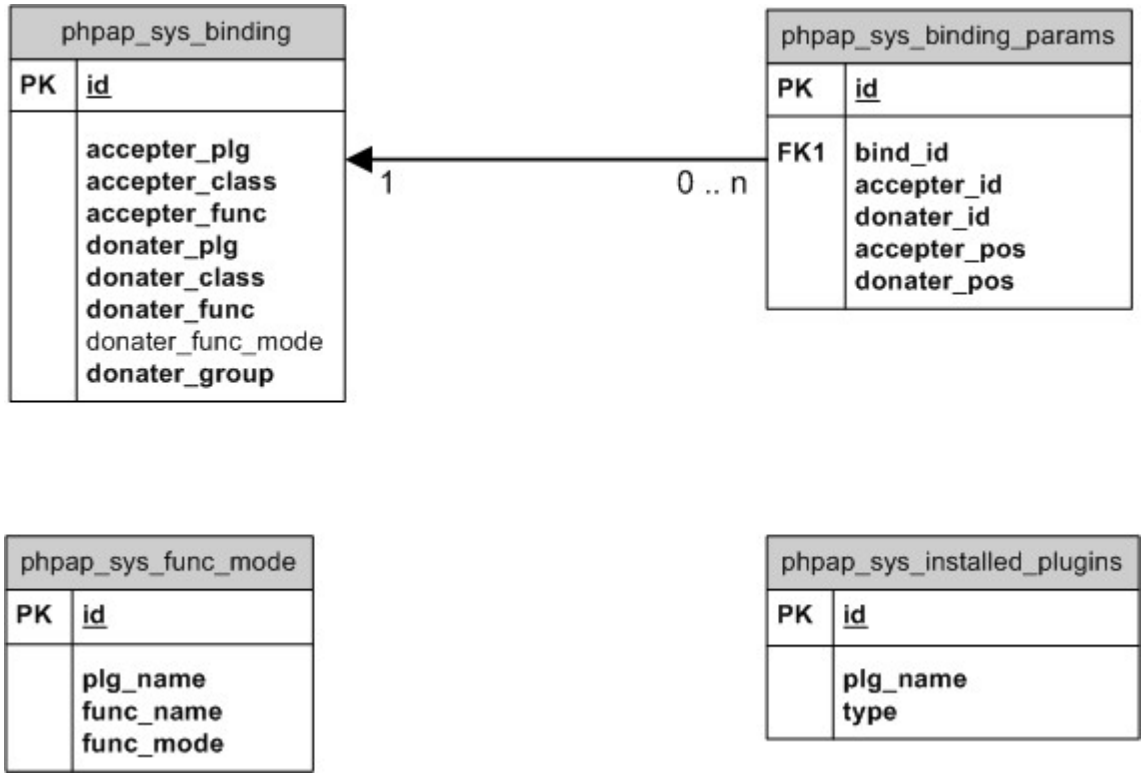
8 Modely PHPAP

8.1 ERD

Modely relací entit jsou poměrně jednoduché, neboť na úrovni potřeb systému nebylo nutné navrhovat žádné složité vazby mezi entitami. Databáze na této úrovni slouží jako přehledný, rychlý a lehce implementovatelný způsob uložení informací proměnného charakteru.

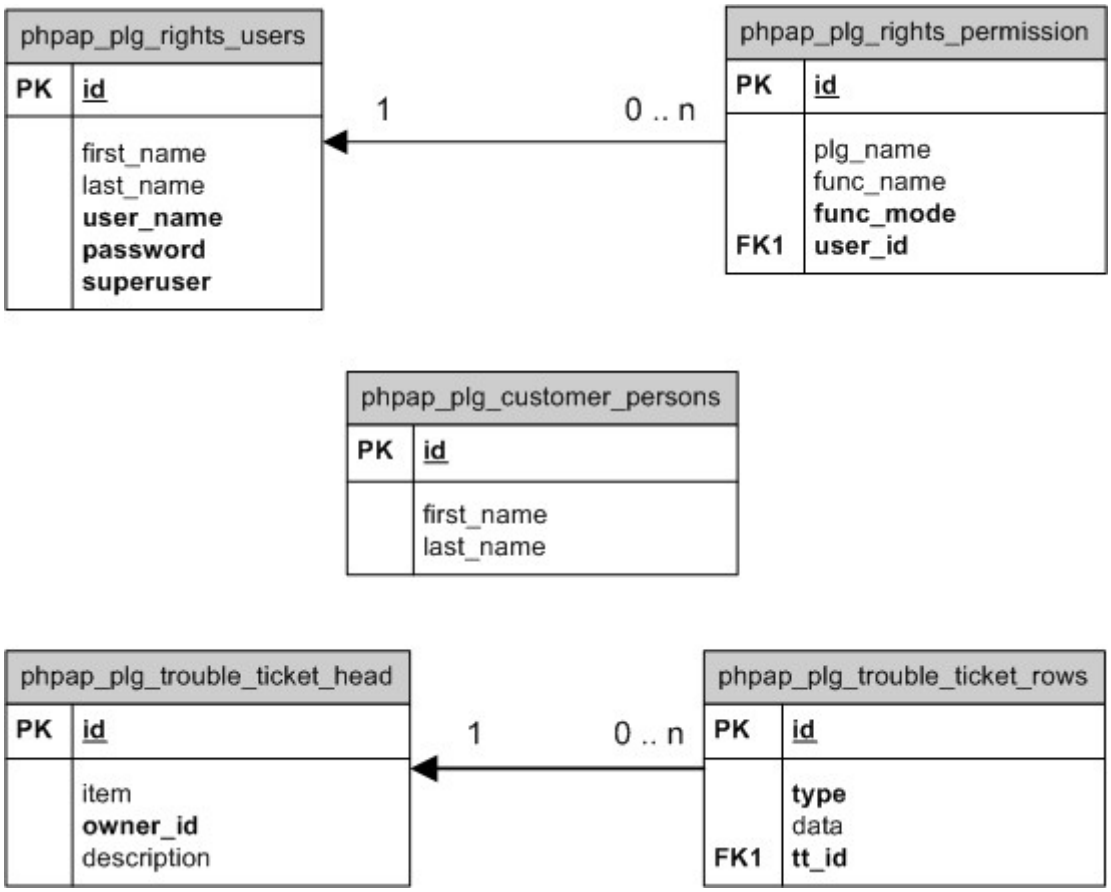
Variantou k databázovému přístupu řešení tohoto problému, je uložení informací do souborů, ve kterých by byly Informace popsány za pomocí jazyka XML.

Entity jádra systému obsahují data o vzájemných vazbách zásuvných modulů, o nainstalovaných modulech a o módech spuštění jednotlivých funkcí (viz obrázek 8.1).



Obrázek 8.1 ER diagram entit jádra PHP aplikační platformy

Entitně relační diagram pro část systému mimo jádro, tedy část zásuvných modulů jsou na obrázku 8.2.



Obrázek 8.2 ER diagram entit zásuvných modulů PHP aplikační platformy

8.2 plg_jméno_pluginu.xml

Příklad souboru, který popisuje zásuvný modul. *PHPAP instalátor* díky tomuto popisu bude schopen správně zavést soubory na správné místo. *PHPAP konfigurátor* využívá tohoto souboru pro načtení informací o funkcích, které lze nastavovat.

```
<?xml version="1.0" encoding="iso-8859-2"?>
<plugin type="plg">
  <name>Jméno Zásuvného Modulu</name>
  <id>plg_identifikator_zasuvneho_modulu</id>
  <author>PHPAP Project</author>
  <creationDate>2008-01-01</creationDate>
  <version>1.0</version>
  <description>Popis zásuvného modulu</description>
```

První částí je hlavička. Je zde uvedena verze XML standardu, kterému soubor odpovídá a kódování, ve kterém je uložen. Element `<plugin>` uvozuje informace vztahující se k zásuvnému modulu, což potvrzuje hodnota plg atributu type.

Dále jsou uvedeny informace o jménu zásuvného modulu, jeho jednoznačném identifikátoru, jménu autora, datu vytvoření, verzi a souhrnný popis funkčnosti.

```
<files>
  <fileName>soubor_jedna.php</fileName>
  <fileName>adresar/soubor_dva.php</fileName>
</files>
```

Sekce files nese informace především pro instalátor zásuvného modulu. Je zde obsažen seznam všech souborů, které jsou nutnou součástí modulu včetně adresářových cest, přičemž za kořen adresářového stromu je brán adresář s názvem zásuvného modulu.

```
<exec_function_group>
  <function name="funkce_jedna">
    <options>
      <option value="1" pos="0">Popis spínače funkcionality na
        pozici 0</option>
      <option value="1" pos="0">Popis spínače funkcionality na
        pozici 1</option>
    </options>
  </function>
</exec_function_group>
```

Další část, viz výše, poskytuje informace konfigurátoru zásuvného modulu. Jsou zde vypsány funkce, jejichž nastavení se dá modifikovat, a možnosti, jakým způsobem může být jejich funkčnost změněna. Element `<function>` uvozuje jednotlivé funkce, které mají různá nastavení `<option>`. Atribut `value` udává výchozí hodnotu, vychází se z množiny obsahující prvky 0 a 1. Atribut `pos` určuje pozici hodnoty v řetězci, který udává mód spuštění funkce.

```
<acceptor_function_groups>
  <function_group name="skupina_mist_pro_akceptaci_funkcí_jedna"
    human_name="Skupina Míst Pro Akceptaci Funkcí Jedna">
    <function name="misto_akceptovani_externi_funkce_jedna"
      class="PHPAPc_plg_identifikator_zasuvneho_modulu">
```

```
      <params>
        <param name="parametr_jedna" type="integer"
          position="0">Popis významu parametru jedna</param>
        <param name="parametr_dva" type="integer"
          position="1">Popis významu parametru dva</param>
      </params>
      <description>Popis místa pro akceptaci jedna</description>
    </function>
  </function_group>
</acceptor_function_groups>
```

Element <acceptor_function_groups>, vyhrazuje prostor pro definování skupin míst, na která jsou navázány externí funkce z jiných zásuvných modulů. Každá skupina má jednoznačný identifikátor a jméno. Smysl skupiny je ten, aby sdružovala místa, která mají logickou spojitost.

Příkladem může být skupina míst, kde jedno z nich je mezi funkcemi prezentující určité informace a druhé mezi funkcemi upravující tytéž informace. Je zřejmé, že v rámci toho, že se stále jedná o stejné informace, spolu místa souvisejí.

Místa jsou tedy pozice v kódu, kde je volána funkce, jako hodnota proměnné, s určitými parametry. Tato funkce sama o sobě nikde není implementována. Implementaci jí dodá až funkce, která je do daného místa připojena. Připojení znamená, že podle tabulky připojení je do proměnné popisující místo přiřazena hodnota v podobě řetězce. Tento řetězec je jméno externí funkce, které má být na zvoleném místě vykonána.

Místa jsou popsána jménem proměnné, která je volána jako funkce, třídou, ve které se volání odehrává a parametry, které jsou poskytovány místem připojované funkci. Atributy parametrů jsou jejich jména, typy a pozice, přičemž hodnota DOM uzlu parametru obsahuje popis parametru.

```
<donater_function_groups>
  <function_group name="skupina_funkci_pro_export_jedna"
    human_name="Skupina Funkcí Pro Export Jedna">
    <function name="funkce_pro_export_jedna"
      class="PHPAPc_plg_identifikator_zasuvneho_modulu">
      <params>
        <param name="parametr_jedna" type="integer"
          position="0">Popis významu parametru jedna</param>
        <param name="parametr_dva" type="integer"
          position="1">Popis významu parametru dva</param>
      </params>
```

```
        <options>
          <option value="1" pos="0">Popis spínače funkcionality na
            pozici 0</option>
          <option value="1" pos="0">Popis spínače funkcionality na
            pozici 1</option>
        </options>
        <description>Popis funkce pro export jedna</description>
      </function>
    </function_group>
  </donater_function_groups>
</plugin>
```

V poslední části vzorového souboru je oddíl popisující funkce zásuvného modulu, které mohou být exportovány, tedy připojeny k jiným zásuvným modulů. Pro zachování kompatibility s místy, do kterých se funkce připojují, jsou i zde zavedeny skupiny, které sdružují funkce podle logických souvislostí. Současně je možné tento koncept ze strany externích funkcí přerušit. Je totiž možné do různých míst připojit tutéž funkci, ale v jiném módu.

Popis externích funkcí spojuje vlastnosti popisu konfigurace funkcí a popisu míst. Tedy popis odpovídá popisu míst, ale navíc je rozšířen o informace, vedoucí k možnosti nastavení módu externě připojované funkce.

9 Struktura systému

Prototypovým řešením problematiky PHP aplikační platformy se stal systém složený z aplikace a k ní navázaných pravidel. Aplikace je souborem skriptů, které zajišťují běh zásuvných modulů. Zásuvné modulu jsou pak psány podle pravidel tak, aby mohli být zpracovány těmito skripty.

Stěžejními osmi třídami jsou `PHPAPc_config_control`, `PHPAPc_core_control`, `PHPAPc_install_control`, `PHPAPc_languages_control`, `PHPAPc_libraries_control`, `PHPAPc_plugins_control`, `PHPAPc_system_control`, `PHPAPc_template_control`. Od těchto tříd jsou vytvořeny instance, které jsou globálně dostupné a poskytují data a metody nezbytné pro běh systému. Tyto třídy jsou umístěny jednotlivě v souborech v adresáři *core*. Soubory jsou pojmenovány dle proměnné střední části názvu tříd, ke které je připojen řetězec *'.class.php'*. Objekty odvozené ze tříd jsou dostupné přes globální pole proměnných `$GLOBALS['PHPAP']['NÁZEV']`, kde název je, stejně jako u souborů, střední část názvu třídy.

9.1 Objekt CORE

Je nejvýše postaveným objektem ze všech objektů tvořících jádro systému. Je odvozen z třídy `PHPAPc_core_control`. Obsahuje již výše zmíněné statické metody `init_units()` a `load_units()`.

První ze jmenovaných připojí zbylých šest souborů se skripty popisující dané třídy a vytvoří jejich instance, jejichž reference umístí do globálního pole `$GLOBALS['PHPAP']`. V konstruktorech těchto objektů nejsou volány žádné metody. Tento úkol obstarává druhá ze jmenovaných metod, které volá v logickém sledu inicializační metody jednotlivých objektů, v případě jejich existence.

Přesněji jde o metody objektu třídy `PHPAPc_config_control`, které naplní členské proměnné informacemi o základní uživatelské konfiguraci systému. Metody instance třídy `PHPAPc_libraries_control` připojují a inicializují knihovny, tvořící framework systému. Posledním inicializovaným objektem je instance třídy `PHPAPc_plugins_control`, který je hlavním aktérem při běhu zapojených zásuvných modulů.

Poslední metodou objektu `PHPAPc_config_control` je `app_run()`. Tato metoda spouští sekvenci metod objektu *PLUGINS*, která zajistí na základě informací zadaných uživatelem, který zásuvný modul má být zpracován. Následně volá metody vybraného modulu pro zjištění požadovaných informací a jejich zobrazení.

9.2 Objekt CONFIG

Je instancí třídy `PHPAPc_config_control`. Jedná se o objekt, který načte a zpracuje hodnoty konfiguračního souboru *config.ini*, nacházejícího se v adresáři *config*. Tyto hodnoty nastavení jsou pak skrze metody objektu dostupné.

9.3 Objekt INSTALL

Je instancí třídy `PHPAPc_config_control`. V rámci žebříčku priorit dosud není implementován. V praxi je zastoupen pokyny v textové podobě, které popisují, jak systém nastavit, aby byla zaručena správná a plná funkčnost.

Má být implementován jako prvek jádra systému, který bude zajišťovat kontrolu nad správností instalace. Druhou funkcí má být řízení uživatelského prostředí pro instalaci a modifikaci instalace.

9.4 Objekt LANGUAGES

Je objektem třídy `PHPAPc_languages_control`. Poskytuje metody pro lokalizaci jednotlivých částí systému do požadovaného jazyka.

Výchozí nastavení čerpá z nastavení v hlavním konfiguračním souboru.

9.5 Objekt LIBRARIES

Je objektem třídy `PHPAPc_libraries_control`. Objekt poskytuje přes výčtové pole přístup k řídícím objektům různých knihoven a metody, které tyto objekty vytvoří.

Pro každou knihovnu je třeba vytvořit řídící třídu, která výchozí rozhraní knihovny standardizuje na rozhraní PHP aplikační platformy. Tyto řídící třídy jdou uloženy v adresáři *core* v podadresáři *libraries.class*. Název skriptu je pak *libraries_název.class.php* a názvem třídy je *PHPAPc_název*.

9.5.1 Knihovna pro práci s databázemi

Objekt řídící třídy `PHPAPc_libraries_database` začleňuje do systému knihovnu Doctrine. Ta vytváří abstrakci nad databázemi.

Nejprve je připojen hlavní skript knihovny a spuštěna funkce pro automatické načtení všech součástí potřebných k běhu. Metodou `Doctrine_Manager::getInstance()` je vytvořena instance správce připojení.

Pro účely PHP aplikační platformy jsou vytvořena dvě databázová spojení. První z nich je nazváno *sys-připojení* a slouží pro přístup k tabulkám databáze, které obsahují informace uložené jádrem systému, jako jsou záznamy o nainstalovaných zásuvných modulech a jejich interakci.

Druhé připojení je nazváno *plg-připojení*. Je k dispozici pro potřebu zásuvných modulů resp. jejich přístupu k databázi.

K vytvoření více spojení mě vedlo několik důvodů. Prvním z nich je demonstrace možnosti jejich vytvoření. Programátor zásuvných modulů není vázán pouze na tato dvě spojení. Pokud je třeba je možné dynamicky vytvořit další.

Hlavním důvodem zůstává možnost výběru databáze podle požadavků, které jsou na ni kladeny. Doporučuji ponechat výchozí nastavení systémového připojení, které je navázáno na databázi SQLite. PHP od verze 5.2. obsahuje rozšíření v podobě knihovny, která implementuje systém řízení báze dat pro databázi SQLite. Samotná data jsou uložena v jediném souboru v adresářovém stromu interpretu PHP. Výhodou tohoto nastavení systémového připojení je, že po instalaci a správném nastavení PHP je databáze okamžitě dostupná bez nutnosti instalace jakýchkoliv jiných databázových serverů. Náročnost na databázi, týkající se uložení a struktury dat vztahující se k informacím o systému, je minimální, proto vyhovuje i řešení, které není založeno na databázi serverového typu.

Další krokem k propojení databáze se systémem je připojení tříd, které za pomocí objektově orientovaného paradigmatu popisují definici dat ať už relačních nebo objektových databází.

Posledním krokem připojení knihovny Doctrine je test, založený na zkušebním dotazu na testovací tabulku.

9.6 Objekt PLUGINS

Je objektem třídy `PHPAPc_plugins_control`. Tento objekt drží kontrolu nad objekty odvozených od řídících tříd jednotlivých zásuvných modulů. Poskytuje metody pro jejich načtení, zjištění metadat ale především pro jejich běh.

Základní metodou je `load_plugins()`, která načte hlavní soubory zásuvných modulů, vytvoří objekty z řídících tříd, a uloží je do globálně dostupného pole.

Metody `plg_select()` a `act_select()` na základě požadavku od uživatele získají informace o tom, který zásuvný modul má být spuštěn a v jakém režimu. Režimem je zde míněna činnost, kterou má zásuvný modul vykonat. Pokud nejsou tyto informace zadány, je v případě zásuvného modulu vybrán první ze seznamu nainstalovaných, jinak skript skončí. V případě režimu zásuvného modulu je při neúspěchu jeho identifikace nastaven výchozí režim, který každý řídící objekt zásuvného modulu je schopen poskytnout.

Metody `get_plg_ops()` a `load_sel_plg_parts()` jsou povinně implementovány v každém zásuvném modulu. Metoda `get_plg_ops()` plní pole akcí, které má definovaný název `$this->ops` a je členskou proměnnou každého modulu. Příklad metody `get_plg_ops()`:

```
function get_plg_ops () {
  $this -> ops[ 'overview' ][ 'm_cu_list' ] = 'm_cu_list';
  $this -> ops[ 'overview' ][ 'v_cu_list' ] = 'v_cu_list';
}
```

Klíč prvního rozměru pole je název režimu, ve kterém zásuvný modul bude běžet. Klíč druhého rozměru pole a zároveň i hodnota takto adresované proměnné je název funkce, která má být vykonána v daném režimu.

Metoda load_`sel_plg_parts()` je zkonstruována za účelem připojení souborů se skripty, které mají být ve zvoleném režimu provedeny. Jedná se o úsporu paměti, tak aby v ní byly vždy načteny pouze skripty, které jsou opravdu spouštěny. Na metodu je kladeno omezení požadující vytvoření dvou objektů `$this->model_obj` a `$this->view_obj`. První jmenovaný reprezentuje metody k získání dat a druhý k jejich vizuální prezentaci.

Další metoda, nazvaná `get_plg_act_ops()`, vybere ze všech funkcí v poli `$this->ops` pouze ty které jsou aktuální vzhledem k požadovanému režimu zásuvného modulu.

O sumarizaci informací týkajících se propojení jednotlivých zásuvných modulů se stará metoda `get_conn_services()`. Na základě informací uložených v systémových tabulkách PHP aplikační platformy vytvoří pole, které nese informace o tom, která z funkcí dárce je spuštěna v bodě příjemce a v jakém módu.

Metoda `get_plg_act_ops_modes()` navazuje na informace o funkcích, které mají být provedeny, a přidává k nim mód spuštění. Mód funkce je řetězec jedniček a nul, které v závislosti na pozici v řetězci, které spouštějí nebo vypínají určenou vlastnost. Ve funkci je možné použít metodu objektu *PLUGINS* k zjištění z módu funkce, zda je vlastnost aktivována či ne, na základě jejího identifikování pro danou funkci unikátním číslem. Toto číslo udává pozici v řetězci módu.

Daemoni jsou funkce, které provedeny při každém běhu systému nezávisle na spouštěném zásuvném modulu. Jsou implementovány jako jejich součást. Jejich využitelnost se vztahuje k implementaci vlastností, jakými jsou například práva, sledování akcí provedených v systému, sledováni statistik a jiných. K načtení daemonů slouží metoda `get_daemons($type = 'begining')`. Proměnná `$type` udává prozatím dvě možnosti. První z nich je *beginning*, kdy jsou načteny funkce, které mají být spuštěny před prvedením vlastních funkcí zásuvného modulu, a druhá *ending*, která má stejný význam, ovšem vše je provedeno až po spuštění vlastních funkcí zásuvného modulu.

Další metoda `run_daemons()`, prochází pole s načtenými daemony a postupně je provádí:

```
function run_daemons()
{
  foreach ( $this -> loaded_daemons as $key_plg => $current_plg )
  {
    if ( isset ( $current_plg ) )
      foreach ( $current_plg as $current_daemon )
      {
        if ( isset ( $current_daemon ) && is_object( $this ->
             loaded_plugins[ $key_plg ] -> daemon))

          $this -> loaded_plugins[ $key_plg ] -> daemon ->
            $current_daemon ();
      }
    }
  }
```

Pro samotné provedení funkcí slouží metody `run_sel_mop($current_op)` a `run_sel_op($current_op)`, kde daný parametr je název funkce která má být spuštěna. První z výše jmenovaných spouští funkce z kategorie model, druhá pak z kategorie view. Automatické vykonání všech funkcí z exekučního pole pak slouží metoda `run_sel_mops()` respektive `run_sel_ops()`.

```
function run_sel_op ( $current_op )
{
  $model = $this -> loaded_plugins[ $this -> plg ] -> model_obj;
  $view  = $this -> loaded_plugins[ $this -> plg ] -> view_obj;
  if ( $this -> sel_ops_modes[ $current_op ] != -1 )
  {
    if ( isset ( $current_op ) && $current_op != '' && is_object (
         $model ) && method_exists ( $model, $current_op ))
      $model -> $current_op ( $this -> sel_ops_modes[ $current_op]);

    if ( isset ( $current_op ) && $current_op != '' && is_object (
         $view ) && method_exists ( $view, $current_op ) )
      $view -> $current_op ( $this -> sel_ops_modes[ $current_op ]);
```

```
    }
}
```

9.7 Objekt SYSTEM

Je instancí třídy `PHPAPc_system_control`. Dosud není implementován. Hlavní funkcí objektu má být systematické zpracování chyb. S tím souvisí i zajištění bezpečnosti skriptů a detekce jejího narušení.

9.8 Objekt TEMPLATES

Je instancí třídy `PHPAPc_templates_control`. Obsahuje metody pro správu nastavení vzhledu aplikací. Měl by být děděn tento objekt řídícími objekty nebo alespoň objekty obsahujícími metody pro zobrazení zásuvných modulů, neboť obsahuje strom metod pro zobrazení webové stránky.

Při zobrazování informací lze pak jednoduše vybírat, do jaké míry bude necháno na objektu *TEMPLATES* a případných metod volaných před samotnou zobrazovací metodou zásuvného modulu. Díky dědění má možnost zobrazovací metoda upravit veškeré aspekty zobrazení. Pro zobrazení webové stránky musí být vždy volána metoda `v_page()` objektu *TEMPLATES*. Ta volá stromově podřízené metody, které postupně zobrazí výsledek.

10 Běh aplikace

Pro každého programátora zásuvných modulů pro PHP aplikační platformu je důležité pochopit jakým způsobem je celý systém spuštěn a které operace jsou provedeny.

Spouštění všech aplikací sestavených ze zásuvných modulů nad PHP aplikační platformou je směrován přes skript *index.php*, který je umístěn v kořenovém adresáři systému. Tento skript zajišťuje přiřazení hodnot proměnným, které udržují informace o prostředí, ve kterém systém běží. Jedná se o základní proměnné s informacemi o absolutních cestách k jednotlivým skupinám skriptů a kompletní adrese serveru, na kterém systém běží. Jsou zde volány statické metody třídy `PHPAPc_core_control`. Jedná se o `load_units()`, zajišťující připojení stěžejních skriptů, a `init_units()`, která inicializuje hlavní objekty systému. Následně je volána statická metoda, která zahajuje rutinu vedoucí ke spouští sestavené aplikace.

Běh systému je možné logicky rozdělit na dvě části, které se vzájemně prolínají. První z nich je běh jádra systému a druhá je pak běh konkrétního zásuvného modulu. Běh zásuvného modulu je přitom podřízen běhu jádra systému.

10.1 Zjištění aktivního pluginu a jeho režimu

Nejprve je pomocí analýzy webové adresy zjištěn identifikátor spouštěného zásuvného modulu, který je uložen v proměnné `plg`, a jeho režimu, uloženého v proměnné `act`.

Je zkontrolována platnost obsahu proměnných. Při neplatnosti hodnoty proměnné `act` je nastaven výchozí režim. Pokud je neplatná hodnota proměnné `plg`, je v databázi instalovaných zásuvných modulů vybrán a nastaven výchozí.

Tyto operace jsou zprostředkovány jádrem systému.

10.2 Nastavení režimu zásuvného modulu

V této části se částečně předává řízení zásuvnému modulu a pomocí jádrem definovaných metod, které jsou implementovány v modulu, se nastavuje systém pro spuštění modulu. Konkrétně se jedná o připojení souborů se skripty a zaregistrování funkcí do exekučního pole podle vybraného režimu.

Dále se provede načtení informací o připojených externích funkcích. Ke každé této funkci je zjištěna konfigurace jejích parametrů a doplnění módu spuštění.

10.3 První cyklus deamonů

Dalším krokem běhu je zjištění aktivních deamonů v systému a jejich provedení. Zjištěním je myšleno prohledání všech nainstalovaných zásuvných modulů, které podají informaci, zda obsahují implementovaný daemon.

V prvním cyklu jsou spouštěny deamoni s příznakem *begining*. Jde o funkce, které mají být provedeny před spuštěním zásuvného modulu. Na základě získaných informací modifikují nastavení systému.

10.4 Spuštění zásuvného modulu

Nejprve jsou provedeny všechny funkce zásuvného modelu z kategorie model. Všechny řetězce, nesoucí název funkcí určených k provedení, v *exekučním poli* jsou testovány, zda existují jako metody objektu model. Pokud ano jsou provedeny. Tím se naplní strukturované proměnné informacemi určenými pro zobrazení. Následuje volání zobrazovacích funkcí.

Systém se snaží volat hlavní zobrazovací funkci z několika míst. Nejprve zkouší volání `v_page()` jako metody objektu model daného zásuvného modulu. Tím je vše, co je zobrazeno, pouze v režii zobrazovacích metod právě tohoto modulu, v závislosti na jeho režimu.

Pokud identifikace režimu selže a objekt reprezentující model není dostupný, snaží se systém volat `v_page()` jako metodu řídícího objektu daného zásuvného modulu.

V případě selhání obou předchozích volání je `v_page()` zavolána přímo jako metoda objektu *TEMPLATES*, kde je definována a odkud je děděna.

To, ve které části jsou spuštěny zobrazovací metody modulu, záleží na místě, ze kterého je volána metoda objektu *PLUGINS* `run_sel_ops()`.

10.5 Druhý cyklus deamonů

Po vykonání všech akcí souvisejících se zásuvnými moduly se systém dostává do poslední fáze, ve které jsou opět spuštěni daemoni, tentokrát s parametrem *ending*.

V praxi se jedná o funkce, které sbírají různé informace poskytnuté zásuvnými moduly o průběhu jejich spuštění resp. jeho výsledcích. Daemoni tyto informace zpracují podle jejich účelu.

Pro lepší pochopení je možné uvést příklad zásuvného modulu, který by měl za úkol monitorovat systém z hlediska provedených akcí, tyto akce archivovat a umožnit jejich prohlížení. Daemon toho zásuvného modulu pak při každém běhu převezme v definovaném formátu informace od zásuvných modulů a po jejich zpracování je uloží do databáze.

11 Tvorba zásuvného modulu

Aby byl dodržen smysl aplikační platformy, je kromě využívání jejích možností také nutné dodržovat určitá stanovená pravidla.

Zásuvný modul, srozuměno jako balík objektů sdružených za účelem správy vyššího logického celku, je umístěn v adresáři plugins. Jméno jeho adresáře je *plg_jméno_pluginu*, kde *plg_* je povinná předpona a *jméno_pluginu* je unikátní identifikátor.

V tomto adresáři je přítomný soubor *plg_jméno_pluginu.xml*, který pomoci jazyka XML a definovaných elementů popisuje zásuvný modul. Může zde být umístěn i soubor *config.ini*, který uchovává případná další nastavení, která jsou zásuvnému modulu vlastní. Posledním souborem, jehož umístění je vyžadováno v tomto adresáři je *index.php*.

Zásuvné moduly jsou chápány jako jednotky postavené na architektonickém vzoru MVC. Soubor *index.php* pak reprezentuje řadič. Soubory v adresáři *model* obsahují třídy pro načtení a připravení dat k zobrazení. Adresář *wiev* sdružuje třídy pro vizuální prezentaci dat.

11.1 Index.php

Je řadičem MVC architektonického vzoru. Obsahuje řídící třídu, která dědí `PHPAPc_templates_control` a implementuje `PHPAPi_plugins_services`. V konstruktoru musí být proměnná `$this->plg_name`, která obsahuje identifikátor třídy. Ten je stejný jako název adresáře, ve kterém je index.php umístěn. Dalšími povinnými proměnnými jsou `$this->plg_human_name`, nesoucí jméno zásuvného modulu, a `$this->default_act`, obsahující jméno výchozího režimu zásuvného modulu.

Je vyžadována implementace metod `get_plg_ops()`, ve které naplní pole `$this->ops`, `load_sel_plg_parts()`, `load_sel_plg_parts()`. Metody jsou blíže popsané v kapitole 9.1.6.

11.2 Soubor s popisem zásuvného modulu

Jedná se o soubor *plg_jméno_pluginu.xml*. Definice souboru je v kapitole 8.2. Volitelně obsahuje párové elementy `<exec_function_group>`, který uvozuje popis funkcí, jež jsou nastavitelné, `<acceptor_function_groups>` a `<donater_function_groups>`, které uvozuje informace o místech, kde jsou funkce připojovány, resp. o funkcích které jsou nabízeny jako připojitelné.

12 Tvorba aplikace

Vlastní vytváření aplikace je práce na uživatelské úrovni, ovšem se znalostmi programátora. Snaha je o minimalizace nutnosti zasahování do zdrojového kódu.

Nejprve je nutné nahrát, dle pravidel PHP aplikační platformy vytvořený, zásuvný modul do adresáře *PLUGINS*. Aby byl pro systém takto nakopírovaný modul viditelný jako nainstalovaný, musí být jeho jméno zaznamenáno do tabulky nainstalovaných zásuvných modulů.

Spuštění konfiguračního nástroje proběhne po zadání následující adresy:

```
http://doména/cesta/k/phpap/system/index.php
```

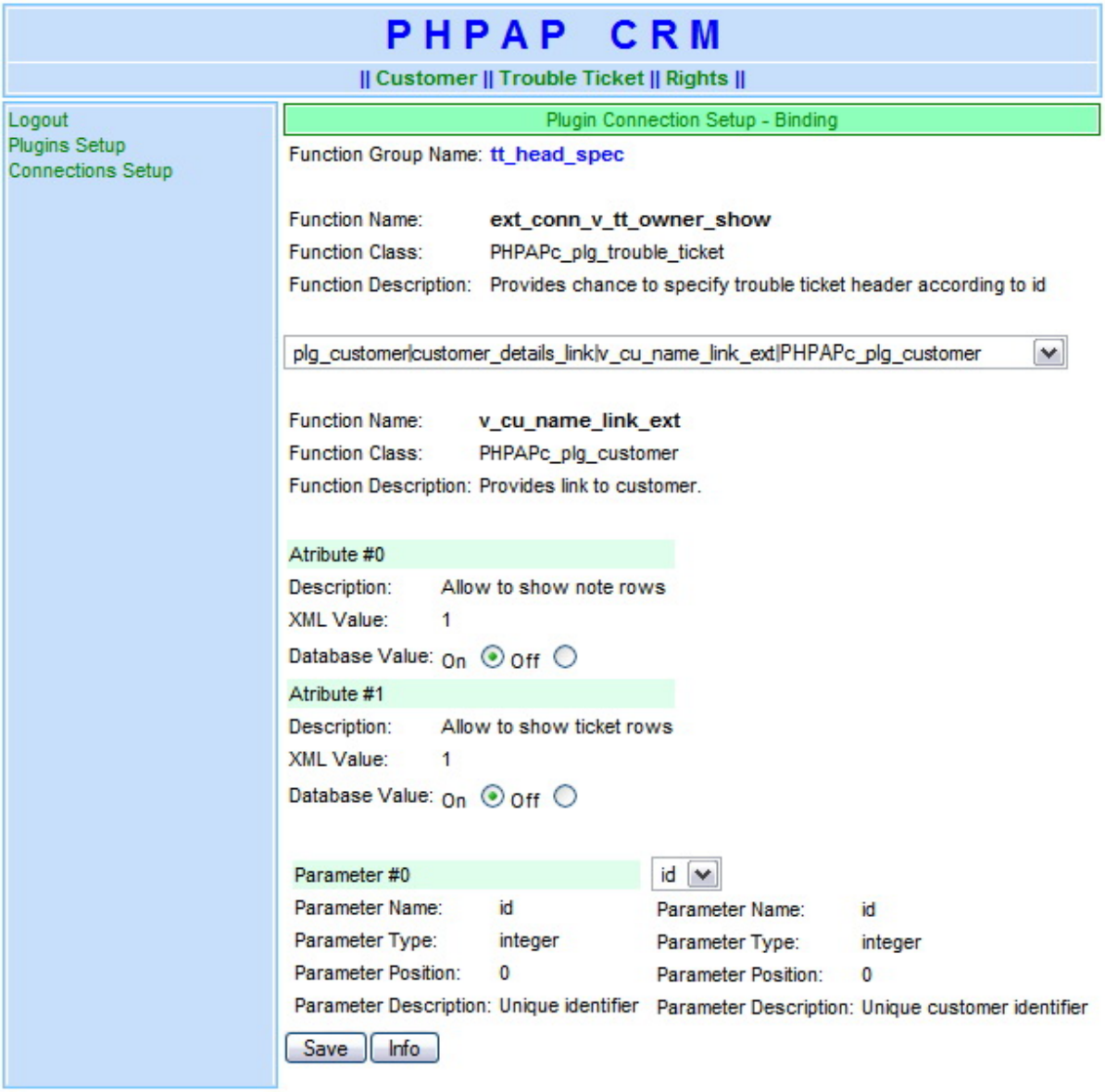
Následně bude uživatel přesměrován na webovou stránku pro ověření přístupových práv. Možnost konfigurace systému je chráněna jménem a heslem. Ty jsou změnitelné v souboru *config.ini*. Pokud je nainstalován modul *Rights*, je nejprve nutné přihlášení se do systému jako jeho uživatel. Teprve poté je připuštěn k ověření přístupu konfigurace systému.

12.1 Konfigurace propojení zásuvných modulů

Na obrázku 12.1 je zobrazen editační režim propojení zásuvných modulů. V záhlaví je globální nabídka nainstalovaných zásuvných modulů, v levé části se nachází lokální nabídka jednoho ze zvolených. V pravé části je pak hlavní obsah určený volbou modulu a jeho režimu. Toto rozložení slouží pouze pro ukázku a záleží pouze na volbě programátora.

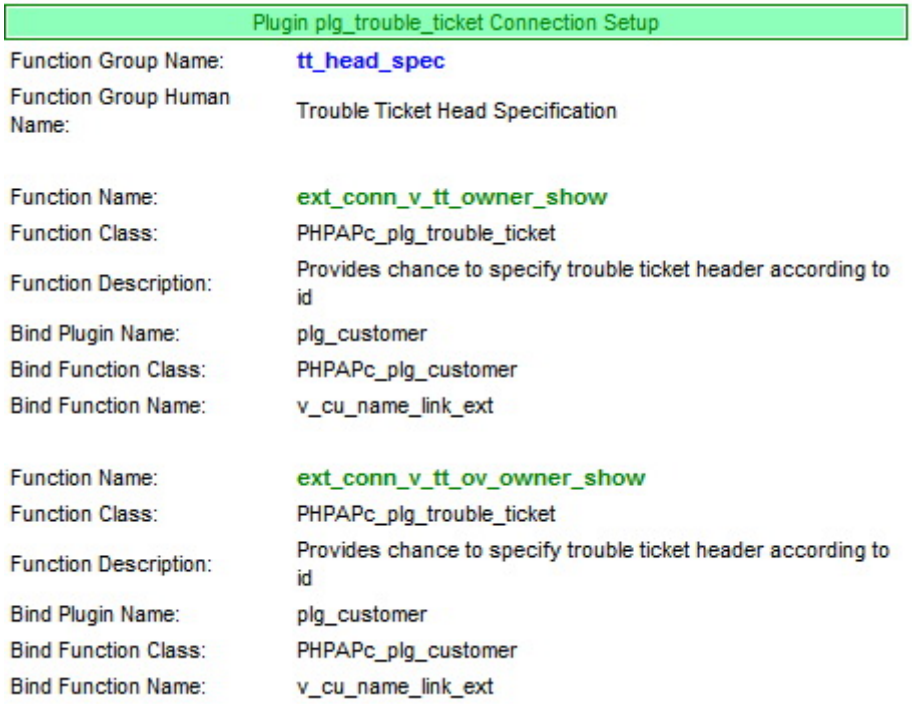
V horní části hlavního bloku jsou informace o místě, do kterého bude připojena externí funkce. Následuje roletka s výběrem funkcí k připojení. Po výběru a stisku tlačítka **Info** se zobrazí podrobné informace o připojované funkci a jejích nastaveních.

Pro úspěšné propojení je důležité také vzájemné svázání parametrů. K tomuto účelu slouží tabulka vazeb parametrů nacházející se na konci hlavního bloku. V levé části tabulky jsou vypsány parametry, které jsou nabízeny daným místem připojení. V levé části je také zobrazen výběr možností, kam hodnoty těchto nabízených parametrů budou přiřazeny, tedy do kterých parametrů připojovaných funkcí.



Obrázek 12.1 Správa vazeb mezi zásuvnými moduly

Po spojení všech požadovaných funkcí je možné zkontrolovat nastavení systému v přehledu, který konfigurační modul poskytuje. Ke zvolenému zásuvnému modulu jsou zde vypsány skupiny funkcí a jejich vazby na funkce připojené (viz obrázek 12.2).



Obrázek 12.2 Přehled vazeb mezi zásuvnými moduly

12.2 Konfigurace zásuvných modulů

Výchozí obrazovka odpovídá přehledovému skriptu pro konfiguraci spuštěných funkcí a zobrazuje přehled nainstalovaných zásuvných modulů. U každého modulu je odkaz na detailní zobrazení jeho nastavení.

Detailní zobrazení představuje kombinaci informací načtených z popisového XML souboru a informací uložených v databázi. Uživateli jsou zobrazeny jednotlivé funkce a nastavení jejich atributů. Každá funkce je popsána jménem a globálním statutem, který udává, zda je daná funkce zapnuta či ne, a atributy, u kterých je individuálně popsán jejich význam, výchozí hodnota stanovená v popisovém XML souboru a hodnota uložená v databázi.

Při přechodu do režimu editace je právě možnost měnit nastavení uložená v databázi, která jsou signifikantní pro běh aplikace. Lze vypnout provádění funkce jako celku skrze hodnotu globálního statutu, nebo jejích částí zapínáním a vypínáním jednotlivých atributů.

Na rozdíl od nastavení připojovaných funkcí, je zde možnost úplného vypnutí. Tato volba je pojmenována jako *Function Status*.

Pokud konfigurace zásuvných modulů na základě nastavování módu spouštěných funkcí není dostatečná, je možné dopsat vlastní konfigurační backend daného zásuvného modulu a stanovit způsob k jeho přístupu.

13 PHPAP CRM

Pro potřebu demonstrace funkčnosti platformy vznikla prozatím jednoduchá aplikace na principu CRM. Sestává se ze tří zásuvných modulů.

13.1 Modul Rights

Stará se o správu uživatelů systému. Umožňuje přiřazení různých nastavení funkcí jednotlivým uživatelů, čímž má každý uživatel jiná práva k práci se systémem. Modul tedy přidává do systému úvodní přihlašovací obrazovku, skrze kterou je uživatel identifikován.

Přiřazování práv k funkcím vztaženo k přihlášenému uživateli je příkladem implementace daemona. Ten je spuštěn při každém běhu systému před provedením vybraného zásuvného modulu a upraví mód spouštěných funkcí podle záznamů, které jsou uloženy jako práva uživatele.

Přístup k nastavení práv uživatelů má pouze uživatel s nastaveným příznakem *superuser*.

13.2 Modul Customer

Slouží pro správu zákazníků systému. Pro účel demonstrace je důležitý především tím, že má implementovány funkce pro export. Jedná se o funkci, která na základě přijatého identifikátoru je schopna zobrazit zákazníka tomuto identifikátoru odpovídajícího, a funkci, která nabízí roletku zákazníků se zobrazením toho, který odpovídá dodanému identifikátoru.

13.3 Modul Trouble Ticket

Neboli správa požadavků od zákazníků. Každý takový požadavek je stručně pojmenován a j k němu přiřazen podrobný popis problému. V této hlavičce je dále místo pro externě připojitelnou funkci. Je popsáno jedním parametrem, identifikátorem. V režimu přehledu je na toto místo připojena funkce pro zobrazení zákazníka, kterému požadavek patří. V režimu editace požadavku je navázána funkce pro změnu zákazníka.

Tělo požadavku tvoří řádky neboli záznamy. Opět na ukázku jsou možnosti dvou typů záznamů. Je možné zvolit typ *Ticket Reference*, který dovoluje přidat odkaz na jiný požadavek uložený v systému. Druhou volbou je typ *Note*, který představuje textovou poznámku.

Pro přidání záznamu slouží roletka s návěstím *Select row* type. Po zvolení typu je pomocí technologie AJAX zobrazen formulář pro doplnění informací o záznamu, který je vložen po stisku tlačítka **Add row** (viz obrázek 13.1).

Obrázek 13.1 Správa požadavků od zákazníků

Závěr

Tato technická zpráva dokumentuje vývoj PHP aplikační platformy. Navazuje na semestrální projekt, který ve světle nových zjištění pozměňuje a rozšiřuje o popis řešení v podobě prototypu CRM aplikace vystavěné na PHP aplikační platformě.

Čtenáři jsou přiblíženy veškeré použité technologie tak, aby byl seznámen s jejich historií. V případě technologických novinek jsou přítomny i ukázky použití s vysvětlením jednotlivých částí kódu. Volba technologií je postavena na vizi použití jednak již známých prostředků v kontrastu s jejich nejmodernějšími metodami použití. Jako příklad lze uvést skriptovací jazyk PHP a k němu nový přístup k databázím pomocí PDO za následného využití frameworku Doctrine.

Volba technologií logicky spíše patří až za analýzu požadavků, přesto, z důvodu plynulého přechodu od teorie k popisu praktického řešení, jsou kapitoly prohozeny. Analýza požadavků je obsáhlá a kompletní, přestože pro potřeby vypracování této zprávy postačila jen její část. Celkově slouží jako motivace celého projektu, tedy oproštění se od tvorby dalšího jednoúčelového systému na úkor investice času k implementaci univerzálnějšího řešení.

Zpráva na základě výsledků analýzy uvádí zasvěcení do problematiky tvorby frameworků a aplikačních platforem. Zde je důležitý bod celé práce, kdy je provedena volba architektury celého systému. Slovo volba není úplně přesné, neboť architektura systému vzniká odvozením a doplnění některých vlastností již známých architektur.

Vzniká tak CRM systém postavěný na PHP aplikační platformě. V souladu s faktem že jde o aplikační platformu, jsou definována i pravidla jakým způsobem ji používat. Jsou tím dány restrikce, které správné používání obnáší, ale i prostředky, které nabízí a které umožňují značné zjednodušení práce programátorům zásuvných modulů potažmo takto vytvořených kompletních aplikací.

Celý projekt vznikal agilní technikou vývoje. Smyslem bylo dokázat, že navrhované řešení bude použitelné v rámci stanovené universality. Již teď, při analýze další etapy vývoje je jasné, že některé koncepce budou muset být přizpůsobeny požadavkům. Jedná se o přidání možností při konfiguraci funkcí, nebo větší propracování kooperace daemonů a zásuvných modulů, s čím souvisí definování formátu předávaných zpráv. Pozitivním závěrem ovšem je, že ani tyto nové požadavky nevyžadují žádnou výraznou změnu do hlavní struktury systému, tedy do jeho myšlenky.

Přínosem této práce je především osvojení si nových možností přístupu k programování ať už ve smyslu použití nových technologií nebo ve smyslu přístupu k vývoji jako takovému.

Literatura

[1] WWW stránky. Design Patterns: Model-View-Controller
http://java.sun.com/blueprints/patterns/MVC.html

[2] WWW stránky. Framework - Wikipedia, otevřená encyklopedie
http://cs.wikipedia.org/wiki/Framework

[3] WWW stránky. HMVC: The layered pattern for developing strong client tiers
http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html?page=1

[4] WWW stránky. Presentation-abstraction-control - Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Presentation-abstraction-control

[5] Eden, H. A., Kazman R. Architecture, Design, Implenentation, 2003.
http://www.eden-study.org/articles/2003/icse03.pdf

[6] Gilmore, W. J. Velká kniha PHP5 a MySQL. Zoner Press, 1, 2005.

[7] Sklar, D. PHP5 moduly, rozšíření a akcelerátory. Zoner Press, 1, 2005.

[8] WWW stránky. Ajax: A New Approach to Web Applications.
http://www.adaptivepath.com/ideas/essays/archives/000385.php

[9] SQLite Documentation.
http://www.sqlite.org/docs.html

[10] PHP Doctrine project documentation.
http://www.phpdoctrine.org

Seznam obrázků