VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

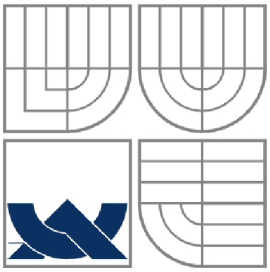
NÁSTROJ PRO USNADNĚNÍ VÝVOJE A TESTOVÁNÍ PHP APLIKACÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

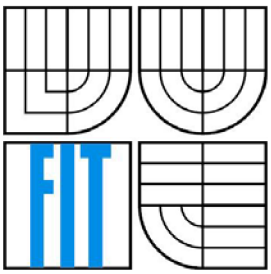
AUTOR PRÁCE
AUTHOR

Bc. JAN PAVELKA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO USNADNĚNÍ VÝVOJE A TESTOVÁNÍ PHP APLIKACÍ

TOOL FOR EASY CODING AND DEBUGGING OF PHP APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAN PAVELKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROMÍR MARUŠINEC, Ph.D., MBA
SUPERVISOR

BRNO 2009

Zde patří originál zadání práce.

Abstrakt

Cílem této diplomové práce je seznámení se s nástroji pro usnadnění vývoje a testování webových PHP aplikací a na základě analýzy obecných požadavků a konkrétních požadavků vývojářů IS VUT v Brně navrhnout a implementovat finální podobu celého nástroje. Práce zahrnuje seznámení se s důležitými pojmy týkající se ladících nástrojů PHP aplikací, analýzu a specifikaci požadavků pro vytvoření nástroje, návrh nástroje a popis jeho implementace včetně zhodnocení výsledků.

Abstract

The goal of this master's thesis is an introducing with tools for easy coding and debugging of PHP applications and to implement such tool on the basis of analysis of common requirements and specific requirements of IS VUT designers. The project includes an introduction with important ideas and terms referring to tool, analysis and specifications of requirements and design and implementation of final tool.

Klíčová slova

Ladící nástroj, PHP, webové aplikace, HTML, JavaScript, AJAX.

Keywords

Debugger, PHP, web applications, HTML, JavaScript, AJAX.

Citace

Jan Pavelka: Nástroj pro usnadnění vývoje a testování PHP aplikací, diplomová práce, Brno, FIT VUT v Brně, 2009

Nástroj pro usnadnění vývoje a testováni PHP aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaromíra Marušince, Ph.D., MBA.
Další informace mi poskytl konzultant práce pan Ing. Michal Jurosz.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Pavelka
25. května 2009

Poděkování

Děkuji tímto vedoucímu diplomové práce, panu Ing. Jaromíru Marušinci, Ph.D., MBA a konzultantovi práce, panu Ing. Michalovi Juroszovi, za vedení mé práce, vstřícnost a podnětné připomínky.

Obsah

Kapitola 1

Úvod

Rozmach internetu udivuje lidstvo den co den. Jsme svědky vzniku neuvěřitelně silného nástroje pro komunikaci, informace, obchod a zábavu. Obrovský nárůst uživatelů internetu po celém světě vede lidi, i vše co společně vytváří, k potřebě být součástí tohoto velmi rozsáhlého informačního zdroje. S tímto nárůstem objemu internetových informací narůstá i počet webových aplikací a lidí, kteří je vytváří. Vznikají nová vývojová prostředí a obecně technologie vývoje webových aplikací, zdokonalují se stávající a roste snaha o dosažení co možná nejlepší efektivity vývoje. A právě zvýšení efektivity vývoje webových aplikací by mělo být hlavním přínosem této práce, a to formou vytvoření nástroje usnadňujícího vývoj a testování těchto aplikací, který je jeden z druhů tzv. ladících nástrojů.

Spousta vývojových prostředí zaměřujících se na webové aplikace má integrovaný nástroj pro ladění, který je ovšem dostupný pouze v rámci onoho vývojového prostředí. Tyto vývojová prostředí potom mohou při ladění kontrolovat běh aplikace, který mají ve vlastní režii. Nelze tak ale např. ladit či sledovat důležité hodnoty aplikace z různých počítačů a míst. Rovněž nelze získat ladící informace přímo ze serveru, na kterém aplikace běží, pokud na něm není spuštěn nějaký speciální nástroj pro tyto účely.

Cílem této práce je vytvoření nástroje, který maximálně využije možností technologií a technik používaných při tvorbě webových aplikací v jazyce PHP k ladění těchto aplikací bez závislosti na vývojovém prostředí. Na rozdíl od vývojových prostředí s integrovaným nástrojem pro ladění, kde ladící nástroj není součástí vytvářené aplikace, v tomto případě ladící nástroj součástí aplikace být musí, což plyne z požadavků a architektury webových aplikací. Jedná se tedy o vytvoření nástroje k usnadnění vývoje a testování PHP aplikací, který lze lehce a snadno začlenit do jakékoliv webové aplikace a pomocí něhož lze sledovat ladící informace přímo za běhu aplikace na serveru. Subjektivnějším cílem je rovněž vyhovět a určitým způsobem tímto dílem přispět vývojářům Centra výpočetních a informačních služeb VUT.

Následující dvě kapitoly se věnují analýze a specifikaci požadavků na vytvoření nástroje. Provedou čtenáře seznámením se s důležitými pojmy práce. První z nich se zaměřuje na porozumění významu některých důležitých pojmů práce, popisuje hlavní technologie, kterých se tato práce týká a uvádí požadavky na výsledný nástroj. Druhá se zaměřuje na specifikaci některých požadavků práce a popisuje problematiku zejména prvků jazyka PHP, které se týkající či souvisí s požadavky.

Čtvrtá kapitola obsahuje seznámení se s již existujícími nástroji pro usnadnění vývoje webových PHP aplikací, které mají podobnou funkčnost jako nástroj této práce. Kapitola popisuje tři základní kategorie existujících nástrojů a uvádí příklady některých jejich zástupců. Tyto kategorie jsou integrovaná vývojová prostředí, moduly a rozšíření PHP pro ladění a třídy a funkce pro ladění.

Následující pátá kapitola se zabývá návrhem nástroje, jeho jednotlivých částí a způsobu fungování. Obsah kapitoly je doplněn některými vývojovými diagramy týkající se návrhu.

Šestá a sedmá kapitola popisuje důležité nebo zajímavé prvky implementace a testování výsledného nástroje spolu s jeho nasazením.

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků a splnění požadavků, zhodnocení z pohledu dalšího vývoje projektu a osobní přínos práce.

Tato diplomová práce navazuje na semestrální projekt, z něhož byla převzata převážná většina částí obsažených v druhé kapitole týkající se analýzy požadavků a rovněž částí čtvrté kapitoly popisující existující nástroje, z nichž některé byly doplněny o další informace.

Programová (implementační) část diplomové práce využívá upravených prototypů nástroje vytvořených v rámci semestrálního projektu.

Kapitola 2

Analýza požadavků

Analýza požadavků je jednou z nejdůležitějších součástí procesu vývoje softwarových nástrojů a obecně všech softwarových produktů. Dobře analyzované požadavky jsou předpokladem k efektivnímu vývoji produktu.

Tato kapitola se zaměřuje na porozumění významu některých důležitých pojmů práce, popisuje hlavní technologie, kterých se tato práce týká a uvádí požadavky na výsledný nástroj.

2.1 Definice pojmů

Nejprve je třeba vysvětlit a definovat hlavní pojmy týkající se této práce. Začněme *webovou aplikací*.

2.1.1 Webová aplikace

Aplikace

Aplikace [1] je programové vybavení počítače (tj. software), které je určeno pro přímou interakci s uživatelem. Účelem aplikace je zpracování a řešení konkrétního problému uživatele a pro interakci s uživatelem má v počítači typicky grafické nebo textové rozhraní. Aplikace se může skládat z několika programů, případně může být několik aplikací spojeno do skupiny, kterou potom označujeme jako *aplikační balíky*. Mezi aplikace nepatří programy, které s uživatelem přímo nekomunikují (např. operační systém).

Webová aplikace

Slovo „webová“ je odvozeno od zkratky WWW neboli *World Wide Web*, což je jedna ze služeb sítě Internet založená na protokolu HTTP. World Wide Web lze definovat jako soustavu propojených hypertextových dokumentů. Hlavními prvky tvořící World Wide Web jsou:

- Soustava hypertextových případně jiných dokumentů a souborů,
- jazyk pro zpracování těchto dokumentů (HTML, XML, aj.),
- přenos v síti zajišťovaný internetovými protokoly (HTTP, FTP, aj.).

Webová aplikace je tedy aplikace fungující nad službou WWW (tj. aplikace internetového protokolu HTTP). Je poskytována uživatelům z webového serveru přes síť Internet, nebo její vnitropodnikovou obdobu (intranet). Klientem je webový prohlížeč, který slouží ke komunikaci s uživatelem, zobrazení výstupu a prvků pro vstup aplikace. Klient ovšem sám o sobě nezná logiku aplikace. Oproti statickým stránkám webová aplikace zahrnuje nástroje pro dynamické generování obsahu, jako je např. CGI, PHP, javovské servlety nebo ASP.

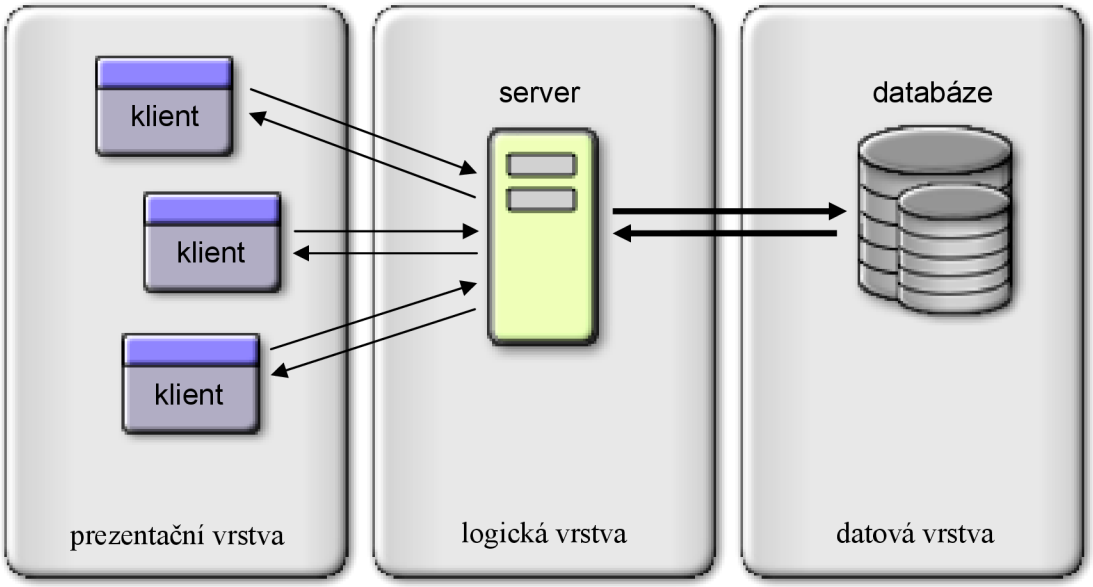
Výhodou webových aplikací [2] je především všudypřítomnost webového prohlížeče jako klienta, díky čemuž lze tyto aplikace aktualizovat a spravovat bez nutnosti šířit a instalovat software na všechny počítače používající aplikaci, kterých může být velmi mnoho.

Rozhraní webové aplikace

Webové rozhraní v některých směrech omezuje funkčnost a možnosti klienta. Metody známé z desktopových aplikací, jako je například vykreslování na obrazovku či obecné techniky jako „drag and drop“, nejsou standardními technologiemi prohlížečů podporovány. Tvůrci webů pro přidání funkčnosti často používají skriptování na straně klienta, zvláště pro vytvoření dojmu interaktivity bez nutnosti znovunačtení stránky, které řada uživatelů shledává rušivým. V poslední době se začínají používat technologie, které umožňují spolupráci skriptů na klientské straně se serverovou částí aplikace. Příkladem takovéto technologie je AJAX, která je popsána dále v podkapitole 2.3.1.

Architektura webové aplikace

Ačkoliv je mnoho možností, webové aplikace jsou obvykle strukturovány jako třívrstvé. V té nejběžnější formě je webový prohlížeč první vrstvou (prezentační), nástroje pro dynamické generování stránek (např. CGI, PHP, javovské servlety nebo ASP) je vrstvou střední (logickou nebo též aplikační) a databáze je vrstvou třetí (datovou). Webový prohlížeč posílá požadavky střední vrstvě, která je obsluhuje prostřednictvím dotazů do databáze (resp. její aktualizací) a generováním uživatelského rozhraní. Grafické znázornění třívrstvé struktury ukazuje obrázek Obrázek 2-1.



Obrázek 2-1: Třívrstvá architektura webové aplikace

Použití

Webové aplikace jsou používány pro implementaci mnoha systémů, jako jsou podnikové a jiné informační systémy, webmaily, internetové obchody, online aukce, diskusní fóra, weblogy, atd. Jednou ze strategií pro softwarové firmy je poskytnout přístup přes web k aplikacím, které byly dříve nabízeny a šířeny jako lokální. Tyto programy umožňují uživatelům platit měsíční či roční poplatek za používání aplikace, aniž by si jej museli nainstalovat na svůj pevný disk.

2.1.2 PHP

PHP (původně *Personal Home Page*, nyní *PHP: Hypertext Preprocessor*) je podle jeho autorů „široce užívaný víceúčelový skriptovací jazyk, který je vhodný zejména pro vývoj webových aplikací a může být vložený přímo do struktury jazyka HTML“ [3].

PHP skripty jsou prováděny na straně serveru, k uživateli je přenášen až výsledek jejich činnosti. Syntaxe jazyka je inspirována několika programovacími jazyky (Perl, C, Pascal a Java). PHP je nezávislý na platformě, skripty fungují bez větších úprav na mnoha různých operačních systémech. Podporuje mnoho knihoven pro různé účely - např. zpracování textu, grafiky, práci se soubory, přístup k většině databázových systémů (mj. MySQL, ODBC, Oracle, PostgreSQL, MSSQL). PHP podporuje celé řady internetových protokolů (HTTP, SMTP, SNMP, FTP, IMAP, POP3, LDAP, …).

V této části práce je jazyk PHP popsán z hlediska vývoje, od jeho začátku až po dnešní podobu. Technické parametry jazyka a další popis pokračuje dále v podkapitole 2.3.1 – Implementační technologie.

Začátky

Vznik jazyka PHP je datován rokem 1994, kdy jeho tvůrce, dánsko-kanadský programátor Rasmus Lerdorf, ve svém volném čase vytvořil jednoduchý systém pro evidování přístupu ke svým stránkám v jazyce Perl. Později autor přepsal systém do jazyka C. Po zdokonalení systému jej uvolnil pod názvem Personal Home Page tools, který se později změnil na Personal Home Page Construction Kit. Postupem času Rasmus Lerdorf vytvořil i nástroj pro používání databáze SQL. Tento program pro zpřístupnění databází na webu nesl název Form Interpreter (FI). Spojením těchto dvou programů vznikl systém PHP-FI.

PHP-FI 2

[4] V roce 1997 byl nástroj PHP/FI přepracován, v této době se však o vývoj staral pouze Rasmus. Později si tohoto nástroje všimli Andi Gutmans a Zeev Suraski, kteří pátrali po jazyce, jenž by jim usnadnil vývoj řešení elektronického obchodování, nezbytného k dokončeni jejich univerzitního projektu.

PHP 3

Zeev a Andi se rozhodli skriptovací jazyk zcela přepracovat. Spojili se s Rasmusem a společně připravili verzi PHP 3. S novou verzí vznikl rovněž nový název – PHP: Hypertext Preprocessor. Autoři chtěli zdůraznit, že PHP je nyní jiným produktem a není určen jen pro osobní potřebu. Autoři také navrhli a implementovali nová rozšíření API. Nové rozhraní API umožňovalo snadnou podporu dalších rozšíření. Ta se dala využít například ke spojení s databází, ke kontrole pravopisu a k dalším procesům. Šlo o technologie, jež jsou středem zájmů mnohých vývojářů, kteří se k projektu připojili, když nebyli členy ústředního vývojového týmu jazyka. Po uvolnění třetí verze PHP v červnu 1998 se odhadovalo, že PHP je instalován na 50 000 domén. Nakonec se dokonce ukázalo, že první verze byla instalována na více než jednom miliónu domén.

PHP 4

Na konci roku 1998 se A. Gutmans a Z. Suraski pustili do dalších úprav. Skriptovací stroj PHP 3 zpracovával skript během čtení. Verze PHP 4 přišla s novým přístupem – „nejprve překlad, pak zpracování“. Skriptovací stroj PHP 4 nepřekládá skript do strojového kódu, ale do bajtového kódu, jenž je pak zpracováván pomocí stroje Zend Engine (název Zend symbolizuje jména **Ze**ev a A**ndi**). Stroj Zend se stal srdcem prostředí PHP 4. Díky novému způsobu zpracování skriptů se výkon jazyka výrazně zvýšil, aniž by došlo k zásadnímu narušení zpětné kompatibility s jazykem PHP 3. Více o Zend Engine je popsáno v podkapitole 3.1.1 této práce.

V jazyce PHP 3 se nižší číslo verze nepoužívalo. Proto byly také všechny verze číslovány jako 3.0.x. To se v PHP 4 změnilo a nižší čísla verzí označují důležité změny v jazyce. Jednotlivé verze a změny s nimi související do dnešní doby jsou:

- **4.0.0** – Přidán pokročilý dvoustupňový systém syntaktické analýzy – Zend engine.
- **4.1.0** – Příchod superglobálních proměnných ($_GET, $_POST, $_SESSION, …)
- **4.2.0** – Vypnutí volby `register_globals` ve výchozím nastavení. Data přijatá prostřednictvím sítě nejsou přímo vkládána do názvů globálních proměnných, uzavírá se tím možnost využití bezpečnostních děr.
- **4.3.0** – Poprvé obsahovala rozhraní příkazového řádku (CLI – Command Line Interface). Dále tato verze přinesla výrazné vylepšení vrstvy souborových a síťových vstupů a výstupů nazývaných proudy (streams).
- **4.4.0** – Dodány některé stránky nápovědy, oprava některých bezpečnostních prvků a chyb. Autoři uvádí tuto verzi jako poslední update PHP 4.

PHP 5

Postupem času se začaly množit požadavky na obecnější objektově orientovaný přístup. Autoři PHP přišli s myšlenkou přepracování objektově orientované části skriptovacího stroje Zend. 13. června 2004 byla představena verze PHP 5, která již stojí na novém Zend Engine II. Kromě podpory pro objektově orientované programování obsahuje PHP 5 velké množství výkonných doplňků. V roce 2008 se PHP 5 stává jedinou stabilní verzí, která se vyvíjí.

2.1.3 Nástroj pro usnadnění vývoje a testování

Tento nástroj lze nazvat jako ladící nástroj (angl. debugger). Typický debugger je počítačový program, který se používá k nalézání a identifikaci chyb v jiných programech. Většinou je možné zobrazit zdrojový kód laděného programu, takže je ihned možné vidět místo, kde se objevila chyba. Většina vývojových prostředí (Integrated Development Environment - IDE) má integrovaný debugger, případně se připojuje na externě spuštěný nezávislý debugger. Používání debuggeru především urychluje implementační část procesu vývoje softwaru. Mezi typické činnosti procesu ladění patří zejména [5]:

- **zarážky (breakpoints)** - možnost zastavit běh kódu na předem daném místě, ať už vybráním řádku kódu nebo při splnění nějaké podmínky
- **krokování** - možnost spouštět jednotlivé kroky programu po částech

- **sledování hodnot proměnných**
- **sledování volání (stack - zásobník) -** tedy lze odpovědět na otázku „která funkce zavolala aktuální funkci“
- případně v průběhu krokování provedení nějakého příkazu zapsaného z konzoly

Jedná se tedy především o zastavení programu v určitém místě. Tato situace může být určité místo programu (výše popsaná zarážka), vyvolání výjimky, čtení nebo zápis do určité proměnné, přístup k hardware apod. Na zastaveném programu je možné zkoumat stav procesu: obsah paměti, hodnoty na zásobníku, objektové závislosti, a z nich usuzovat, proč k chybě v programu dochází.

V případě této práce (tj. webové aplikace v PHP) je ovšem ladící nástroj chápán spíše obecněji jako nástroj, který usnadní vývoj aplikace. Vzhledem ke struktuře a principu fungování webových aplikací nejsou některé činnosti typického debuggeru dost dobře proveditelné, jako např. oddělení ladícího nástroje a aplikace jako dvou nezávislých programů. Ladící nástroj v tomto případě musí být součástí aplikace. Rovněž může být problém s pozastavením programu, zjišťováním obsahu paměti atd. K usnadnění vývoje jistě poslouží prvky jako vypsání aktuálních hodnot proměnných, polí, globálních polí, sledování volání funkcí, doba vykonávání skriptu, atd., čímž se dostáváme k požadavkům na systém.

2.2 Obecné požadavky

Vzhledem k faktu, že nástroj bude sloužit výhradně vývojářům, nemusí pro něj striktně platit standardní požadavky internetových aplikací. Např. skutečnosti jako používání nástroje lidmi s určitým tělesným postižením, jako je nevidomost nebo omezená pohyblivost, nejsou příliš pravděpodobné. Hlavní hlediska obecných požadavků a cílů nástroje jsou následující [6]:

- **Použitelnost**
- **Využitelnost**
- **Přístupnost**
- **Vzhled**
- **Zabezpečení**
- **Rozšiřitelnost**

2.2.1 Použitelnost

Nástroj a jeho ovládání musí být přehledný a srozumitelný, neměl by nutit uživatele přemýšlet nad věcmi, které by měl dělat automaticky či intuitivně. Pokud použije uživatel nástroj poprvé, měl by snadno a rychle pochopit jeho ovládání. Naopak pokud je již se systémem obeznámen, měl by snadno a rychle a bez problémů dosahovat svých plánovaných cílů. Uspořádání a ovládání nástroje by mělo být lehce zapamatovatelné.

Dále je vhodné, aby výstup nástroje, který se bude zobrazovat převážně v internetovém prohlížeči, byl z hlediska HTML kódu validní[1]. Jelikož ovšem bude nástroj sloužit výhradně vývojářům a z hlediska koncového uživatele vyvíjené aplikace nebude viditelný, nemá tento požadavek zásadní prioritu.

Výhodou je rovněž menší velikost (z hlediska objemu dat) nástroje, která může ovlivňovat rychlost zobrazování celkové aplikace. Jedná se především o načítání obrázků, což souvisí s požadavky na vzhled nástroje, popsanými v podkapitole 2.2.4.

2.2.2 Využitelnost

Systém musí být univerzální, použitelný obecně pro různé webové PHP aplikace. Měl by také umožňovat zobrazení různých cizojazyčných znaků ve správném nezkresleném tvaru. Vhodné také je, aby fungoval na většině standardních webových serverech beze změny funkčnosti.

2.2.3 Přístupnost

Dva hlavní důvody, které mohou zapříčinit problémy ze strany uživatele, jsou:

- **Technické důvody** – uživatelé by neměli být znevýhodněni jednak z hlediska softwaru, který ke zobrazení výstupu nástroje používají (tzn. různé typy webových prohlížečů např. Internet Explorer, FireFox, Opera, apod.) a jednak platformy, na které nástroj používají.
- **Důvody tělesného postižení** – zde se jedná především o již zmíněné problémy s tělesným postižením, a to zejména vizuální kontakt s nástrojem.

Na řešení důvodů vážnějších tělesných postižení, které mohou zapříčinit problémy ze strany uživatele, se nemusí klást příliš veliký důraz vzhledem k typu uživatelů, kteří budou nástroj používat.

2.2.4 Vzhled

Vzhled (nebo také design) vytváří první dojem uživatele, proto je pro většinu webových aplikací velmi důležitý. Nástroj této práce ovšem do oné většiny příliš nepatří. Typická webová aplikace je vyvíjena především pro uživatele – návštěvníky internetu, kteří se chtějí dozvědět, zařídit nebo vyřídit nějaké informace. Nástroj pro usnadnění vývoje a testování webových aplikací ovšem slouží vývojářům aplikace a tato cílová skupina ho také bude jako jediná používat. Z pohledu vývojáře je prioritní spíše rychlost a přehlednost než krásný grafický vzhled, který může být někdy až nežádoucí. Prioritou je tedy přehlednost, která převažuje nad pěkným grafickým dojmem uživatele.

Výstup nástroje by měl být vhodně umístěn mimo výstup vyvíjené aplikace. Je tedy žádoucí nemíchat dohromady výstupy vyvíjené aplikace a nástroje.

---

[1] *Validitou* webových aplikací se rozumí jejich soulad s technickými pravidly pro psaní zvoleného značkovacího jazyka, v němž jsou vytvořeny. Tímto jazykem bývá nejčastěji HTML a jeho různé verze. Validita webových aplikací hraje roli v přístupnosti a použitelnosti těchto aplikací, pro fungování aplikace ovšem není zásadní.

2.2.5 Zabezpečení

Oproti vzhledu je zabezpečení naopak velmi důležité. Jelikož bude standardně nástroj používán přes internet jako aplikace klient-server, mechanismy zabezpečení by měly být velmi důsledné. Je třeba docílit, aby nástroj mohly používat výhradně a jen oprávněné osoby.

2.2.6 Rozšiřitelnost

Vzhledem k univerzálnosti, obecnosti a s ohledem na využitelnost nástroje v různých aplikacích by měl být nástroj navržen tak, aby mohl být dle nových požadavků a záměrů případně doplněn o další funkce a chování, které v něm nejsou obsaženy.

K rozšiřitelnosti patří i nutnost možnosti aktualizace systému vzhledem k času, případným novým technologiím a zvyklostem.

2.3 Požadavky vývojářů IS VUT v Brně

2.3.1 Implementační technologie

HTML

HTML (zkratka anglického „Hypertext Markup Language“ – „značkovací jazyk pro hypertext“) je značkovací jazyk pro tvorbu hypertextových dokumentů v prostředí WWW vyvinutý konsorciem W3C (World Wide Web Consortium).

W3C je nezávislá, nezisková standardizační organizace, která produkuje veřejné normy a stará se o jejich propagaci. Každá norma prochází vývojem popisovaným takzvanými *Recommendation track technical reports*. Jsou to zprávy, které procházejí přijímacím procesem, na jehož konci se stávají platnou specifikací. Tyto zprávy procházejí následujícími pěti stádii, v každém získávají konkrétnější a propracovanější podobu až do konce procesu, kdy se stávají specifikací (v přesném překladu „doporučením“) [7]:

1. **Working draft** (pracovní návrh) – tato první veřejná zpráva je čistě pracovní, může v ní dojít k zásadním změnám a není nijak uznána W3C.
2. **Last call working draft** (poslední pracovní návrh) – tato zpráva je již pokročilejším pracovním návrhem, je určena především ke komentování ostatními skupinami W3C, členy W3C a veřejností.
3. **Candidate Recommendation** (kandidát na specifikaci) – tato zpráva je upravena podle komentářů veřejnosti i ostatních skupin a členů W3C. Zároveň v sobě zahrnuje ohled na ostatní dokumenty W3C. Je výzvou ke zkušebnímu zavedení dané technologie, přičemž W3C skupina, která zprávu vydala, je otevřena komentářům a zkušenostem se zavedením technologie.
4. **Proposed Recommendation** (navržená specifikace) – tato zpráva čeká na schválení W3C.

5. **Recommendation** (specifikace, v terminologii W3C doporučení) – tato zpráva je uznána W3C jako platná specifikace určitého standardu a je určena k zavedení a propagaci.

Jazyk HTML [6] prodělal poněkud živelný vývoj završený verzí HTML 4 (poprvé publikována v prosinci 1997, poslední revize pod označením HTML 4.01 je z prosince 1999). Dnes již spíše historický význam má verze HTML 3.2 z ledna 1997.

Nástupcem jazyka HTML je jazyk XHTML verze 1.0 získal od W3C status doporučení (tj. nejvyšší status) v lednu 2000. Účelem i významovým rozsahem se jedná o HTML 4.01, řídí se však syntaktickými pravidly XML. Modularizovaná verze XHTML vstoupila v platnost v květnu 2001 pod označením XHTML 1.1. *Modularizace* přináší možnost vývoje různých XHTML profilů uzpůsobených pro různé typy zařízení a skupiny uživatelů. Příkladem XHTML profilu, vyvinutého pro přenosná zařízení (mobilní telefony, PDA, pagery, atd.), je XHTML Basic z prosince 2000.

Stávající verze XHTML jsou:

XHTML 1.0 – První specifikace, jejíž cílem bylo převedení staršího jazyka HTML tak, aby vyhovoval podmínkám tvorby XML dokumentů a přitom byla zachována zpětná kompatibilita.

XHTML Basic – Příklad minimální sady modulů potřebné k vytvoření XHTML dokumentu, která je cílená na mobilní aplikace.

XHTML Mobile Profile – někdy taky XHTML MP je postaveno na základě XHTML Basic a je určeno pro použití v mobilních telefonech. Někdy je také označováno jako WAP 2.0. XHTML Mobile Profile podporuje na rozdíl od WAP 2.0 barvu a barevné obrázky ve formátech GIF, JPEG a PNG.

XHTML 1.1 – Příklad rozsáhlé sady modulů pro komplexnější tvorbu XHTML dokumentů. Vynechává již prakticky všechny prezentační vlastnosti. Je velice podobné XHTML 1.0 Strict, ale na rozdíl od něj může vzhledem ke své modularizaci sloužit jako základ budoucím rozšířeným dokumentům z rodiny XHTML. Dokumenty tohoto typu mají standardně koncovku .xhtml.

XHTML-Print – Zaměření na tiskový výstup. Tato verze je ve třetím vývojovém stádiu, tzn. Candidate Recommendation.

XHTML 2.0 – Není zamýšleno tak, aby bylo zpětně kompatibilní se svými předchůdci. Je teprve v prvním vývojovém stádiu, které má označení Working Draft.

CSS

CSS je zkratka pro anglický název „Cascading Style Sheets“, česky „ kaskádové styly“. Je to jazyk pro popis způsobu zobrazení stránek napsaných v jazycích HTML, XHTML nebo XML. Technologie CSS byla navržena konsorciem W3C. Doposud byly vydány zatím dvě verze specifikace CSS1 a CSS2 (plus CSS 2.1), a pracuje se na verzi CSS3. Hlavním smyslem je umožnit návrhářům oddělit vzhled dokumentu od jeho struktury a obsahu.

Původně měl toto umožnit už jazyk HTML, ale v důsledku nedostatečných standardů a konkurenčního boje výrobců prohlížečů se nevyvinul jak bylo zamýšleno. Starší verze HTML obsahují celou řadu elementů, které nepopisují obsah a strukturu dokumentu, což je dnes hlavní myšlenkou vývoje, ale i způsob jeho zobrazení. Z hlediska zpracování dokumentů a vyhledávání informací není takovýto vývoj žádoucí. Dalo by se říci, že CSS je jakýmsi rozšířením možností

zobrazení obsahu (X)HTML.

Javascript

JavaScript [8] je multiplatformní, objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape. Zpravidla se používá jako interpretovaný programovací jazyk pro WWW stránky, vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

Na rozdíl od ostatních zmíněných jazyků, patří mezi skriptovací jazyky, jejichž kód je zpracováván a interpretován na straně klienta, což vede k celkovému zrychlení aplikace a snížení zátěže serveru. Nevýhodou je ovšem fakt, že ne každý klient má povoleno tento jazyk na svém prohlížeči spouštět. K tomuto problému ovšem bude při vývoji IS přihlíženo.

Jeho syntaxe patří do rodiny jazyků C/C++/Java. I když se to na první pohled může zdát, s jazykem Java nemá, až na podobný název a podobnou syntaxi, nic společného. JavaScript byl v červenci 1997 standardizován asociací ECMA (European Computer Manufacturers Association) a v srpnu 1998 ISO (International Standards Organization). Standardizovaná verze JavaScriptu je pojmenována jako ECMAScript a z ní byly odvozeny i další implementace, jako je například ActionScript.

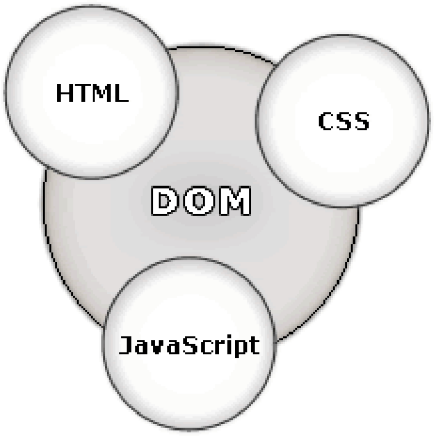
JavaScript obsahuje několik metod, pomocí nichž je možné přistupovat k jednotlivým elementům dokumentu a modifikovat jejich vlastnosti nebo vytvářet elementy nové, díky čemuž lze pracovat s DOM.

DOM

DOM (Document Object Model) je objektově orientovaná reprezentace webových dokumentů umožňující modifikaci obsahu, struktury nebo stylu dokumentu a jeho částí. DOM představuje standardizované programátorské rozhraní pro přístup ke struktuře, stylu a událostem dokumentu. Využívá koncepce OOP (objektově orientované programování), není však vázán na konkrétní programovací jazyk. Tím umožňuje dalším softwarovým aplikacím pohodlně manipulovat s HTML a XML dokumenty a vytvářet tak dynamické webové stránky.

V případě nástroje této práce bude technologie DOM využívána jazykem JavaScript.

Technologie modifikace obsahu dokumentu bez spolupráce se serverovou stranou webové aplikace je nazývána jako DHTML. Tato technologie bude hojně využívána nástrojem této práce. V DHTML je přístupný každý objekt webové stránky a s každým objektem je možné manipulovat – měnit jeho obsah, způsob zobrazení apod. Jako objekty jsou přístupné i kaskádové styly připojené ke stránce. DHTML je skloubením technologií DOM, JavaScript a CSS. Grafické znázornění technologie DHTML ukazuje obrázek Obrázek 2-2.



Obrázek 2-2: Grafické znázornění technologie DHTML

PHP

Krátké seznámení s jazykem PHP a jeho popsání z hlediska vývoje již bylo uvedeno v podkapitole 2.1.2. Nyní je vhodné si popsat jeho technické parametry a další údaje.

PHP je tedy skriptovací jazyk, jehož příkazy jsou do webových stránek (přesněji do HTML kódu) umístěny mezi speciální oddělovače – tagy (`<? ?>` nebo `<?php ?>`). Vlastní skriptovací jazyk kombinuje několik programovacích jazyků – C, Perl, Java, obsahuje rozsáhlé knihovny funkcí pro zpracování textu, grafiky, práci se soubory, přístup k různým databázovým serverům, podporu řady internetových protokolů (HTTP, FTP, POP3, SMTP, ...) a lze ho také jednoduchým způsobem doplnit o funkce nové, které standardně nejsou jeho součástí. Stránky obsahující PHP vsuvky jsou pro odlišení od běžných webových stránek ukládány s příponou .php (.php3, .phtml). Vývoj technologie PHP stále pokračuje.

Velkou výhodou technologie PHP je fakt, že je šířena jako freewarový[2] produkt včetně zdrojových kódů. PHP není závislé ani na platformě ani na webovém serveru, na kterém je provozováno. Lze jej tedy užívat jak např. pod Windows tak i pod jinými operačními systémy s tím, že není svázáno s žádným konkrétním serverem a proto může běžet na libovolném. Dokazuje to velké procento serverů, na kterém je PHP v současnosti používáno a jeho velká podpora ze strany hostingových služeb.

PHP jazyk je velice oblíbený. Jeden z důvodů je jednoduchost jeho použití. Tím, že PHP kombinuje vlastnosti více programovacích jazyků, ponechává vývojáři částečnou svobodu v syntaxi.

AJAX

AJAX (Asynchronous JavaScript and XML) je obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání. Na rozdíl od klasických webových aplikací poskytují uživatelsky příjemnější prostředí. AJAX ve skutečnosti není konkrétní specifická technologie, ale pojem označující použití několika technologií dohromady s určitým cílem [9]. Primární komponentou AJAXU je JavaScript.

Prostřednictvím AJAXu probíhají operace na pozadí. Snahou technologie je zvýšit plynulost aplikace, kde se právě díky načítání na pozadí webové aplikace začínají aplikace blížit desktopovým.

---

[2] Software, který je šířen bezplatně

Za nevýhody je považována změna paradigmatu používání webu, které se liší od konceptu posloupností stránek, jenž je možné procházet tlačítky „Další“ a „Zpět“.

2.3.2 Funkcionální požadavky

Požadované funkce výsledného nástroje jsou shrnuty do následujících bodů:

- **Zobrazení aktuálních hodnot proměnných** – důležitou funkcí nástroje je zobrazení hodnot proměnných různého typu včetně polí, objektů, superglobálních proměnných jako $_SESSION, $_GET, $_POST, $_COOKIE atd.
- **Zobrazení změn hodnot proměnných** – jedná se o funkci zobrazení změn v hodnotách superglobálních proměnných, které nastaly od začátku zpracování skriptu po jeho konec
- **Trasování chodu skriptu** – nástroj by měl nějakou vhodnou technikou umožnit zobrazení uživatelem zadaných bodů ve skriptu a vypsat ladící informace v těchto bodech
- **Sledování volání funkcí** – měla by být možnost sledovat volání funkcí v aplikaci. Uživatel by měl mít možnost zobrazit seznam volaných funkcí, který obsahuje např. název funkce, uplynulý čas, okamžik, kdy se daná funkce zavolala, úroveň vnoření volání funkce, číslo řádku a soubor, kde se funkce vyskytuje atd.
- **Doba vykonávání skriptu** – jedná se o funkci zobrazení doby vykonávání aktuálního skriptu
- **Možnost nastavení parametrů** – nástroj by měl obsahovat měnitelné parametry různého typu, např. pro nastavení oprávnění, vzhledu nástroje, obsahu nástroje atd. Tyto parametry by měly být vhodně dostupné např. v rámci jedné sekce „nastavení“
- **Ukládaní konfigurace nástroje** – je třeba, aby se různá nastavení nástroje ukládala, aby při opětovném spuštění aplikace s nástrojem byl zachován jeho předešlý stav nastavení. Vhodné by bylo ukládání jak lokální, v rámci prohlížeče, např. za použití COOKIES[3], tak globální, v rámci uživatele, který nástroj používá, např. uložením konfigurace do databáze nebo souboru
- **Ukládání konfigurace po názvech** – také je vhodné, aby nástroj umožnil ukládat různé typy konfigurace nastavení nástroje, nejlépe každou konfiguraci pod zadaným názvem. Mezi těmito uloženými konfiguracemi by měla být možnost přepínání z jedné konfigurace na druhou
- **Omezení výpisů** – pokud budou výpisy informací příliš dlouhé, je vhodné tyto určitým způsobem omezit, např. stanovením maximálního počtu řádků výpisu informace, stanovení max. délky výpisu apod.
- **Oznámení a informace o chybách** – pokud nastane chyba v aplikaci, nástroj by měl zobrazit informace o chybě, kde tato chyba nastala, typ chyby atd.

[3] Jako Cookie se v protokolu HTTP označuje malé množství dat, která WWW server pošle prohlížeči, který je uloží na počítači uživatele. Při každé další návštěvě téhož serveru pak prohlížeč tato data posílá zpět serveru. Cookies se běžně ukládají v rámci internetového prohlížeče.

- **Vytváření uživatelských chyb** – pokud chce uživatel explicitně sdělit chybu v kódu nebo v jiném pohledu na aplikaci, měl by mít možnost toto provádět vytvářením speciálních uživatelských bodů v kódu včetně jejich popisu, které nástroj klasifikuje jako uživatelské chyby.
- **Zobrazení informací o aktuálním uživateli** – nástroj by měl identifikovat aktuálního uživatele a vhodným způsobem dát uživateli najevo, kdo nástroj používá.
- **Délka běhu databázových dotazů** – jedná se o zobrazení doby vykonání uživatelem určených databázových dotazů aplikace.

Další požadavky na funkce nástroje jsou s menší prioritou a nemusí být nutně splněny:

- **Seznam všech include a require** – *include* a *require* jsou funkce PHP, které do zdrojového kódu jednoho souboru vloží zdrojový kód souboru jiného. Bylo by vhodné zobrazit např. strom všech takovýchto vložení.
- **Revize aplikací** – zobrazení čísla revize aplikace, pokud aplikace používá systém revizí
- **Výpis z error_logu a access_logu Apache** – *error_log* je soubor, do kterého server Apache zapisuje chyby (errors), varování (warnings) a poznámky (notices) aplikace a *access_log* je soubor, do kterého server Apache zapisuje historii jednotlivých HTTP požadavků při používání aplikace. Bylo by vhodné umožnit zobrazení výpisů z těchto dvou souborů.

Některé z požadavků na funkci nástroje jsou dále rozebrány v podkapitole 3.5.

2.3.3 Požadavky na místo působnosti

Kromě vytvoření univerzálního nástroje, který může být použit obecně v různých webových PHP aplikacích je třeba také nástroj integrovat do aktuální aplikace IS VUT v Brně.

Také by bylo vhodné nástroj uvolnit jako open source, tzn. zpřístupnit nástroj bezplatně široké veřejnosti.

2.3.4 Ostatní požadavky

Další požadavky na výsledný nástroj jsou:

- **Vytvoření různých jednoduchých testovacích aplikací pro nástroj** – pro ukázky funkčnosti nástroje by bylo vhodné vytvořit několik jednoduchých testovacích aplikací, na kterých bude předvedeno fungování nástroje.
- **Zpracování uživatelské příručky** – k nástroji je třeba vytvořit uživatelskou příručku (manuál) pro koncového uživatele, která bude obsahovat popis ovládání a možností nástroje.
- **Zpracování dokumentace** – k nástroji je třeba vytvořit dokumentaci aplikačního rozhraní, která bude popisovat třídy a funkce, jejich parametry atd. Vhodné jsou také výstižné komentáře přímo ve zdrojovém kódu nástroje.
- **Kód programu by měl být objektově orientovaný**

Více o zpracování dokumentace a uživatelské příručky je popsáno v podkapitole 3.4.

2.4 Uživatelské úrovně

Uživatele výsledného nástroje lze rozdělit do dvou úrovní podle druhu oprávnění, jejichž hlavním rozlišujícím faktorem je oprávnění přidávat, editovat a mazat uživatele. Jednotlivé uživatelské úrovně jsou následující:

1. **Pokročilý uživatel** – Tato úroveň je nejvyšší úroveň oprávnění. Pokročilý uživatel má možnost využívat všech dostupných funkcionalit nástroje.
2. **Pověřený uživatel** – Uživatel této úrovně má stejné možnosti jako Pokročilý uživatel s výjimkou správy uživatelů nástroje. Na rozdíl od Pokročilého uživatele nemá možnost přidávat, editovat a měnit uživatele.

Uživatelské úrovně a jejich oprávnění jsou přehledně znázorněny v diagramu případů použití v podkapitole 3.6 této práce.

Kapitola 3

Specifikace požadavků

Specifikace požadavků je další z důležitých částí procesu vývoje softwarových nástrojů. Detailně specifikované požadavky jsou předpokladem k efektivnímu vývoji produktu.

Tato kapitola se zaměřuje na specifikaci některých požadavků práce a popisuje problematiku zejména prvků jazyka PHP, které se týkající či souvisí s požadavky.

3.1 Průběh překladu skriptu v PHP

Pro získávání ladících informací z PHP je vhodné vědět, co se děje uvnitř, když se spustí nějaký PHP program a generuje nějakou webovou stránku. Tato podkapitola poskytuje pohled do nitra PHP a vysvětluje, co se děje při vykonávání skriptu PHP [14].

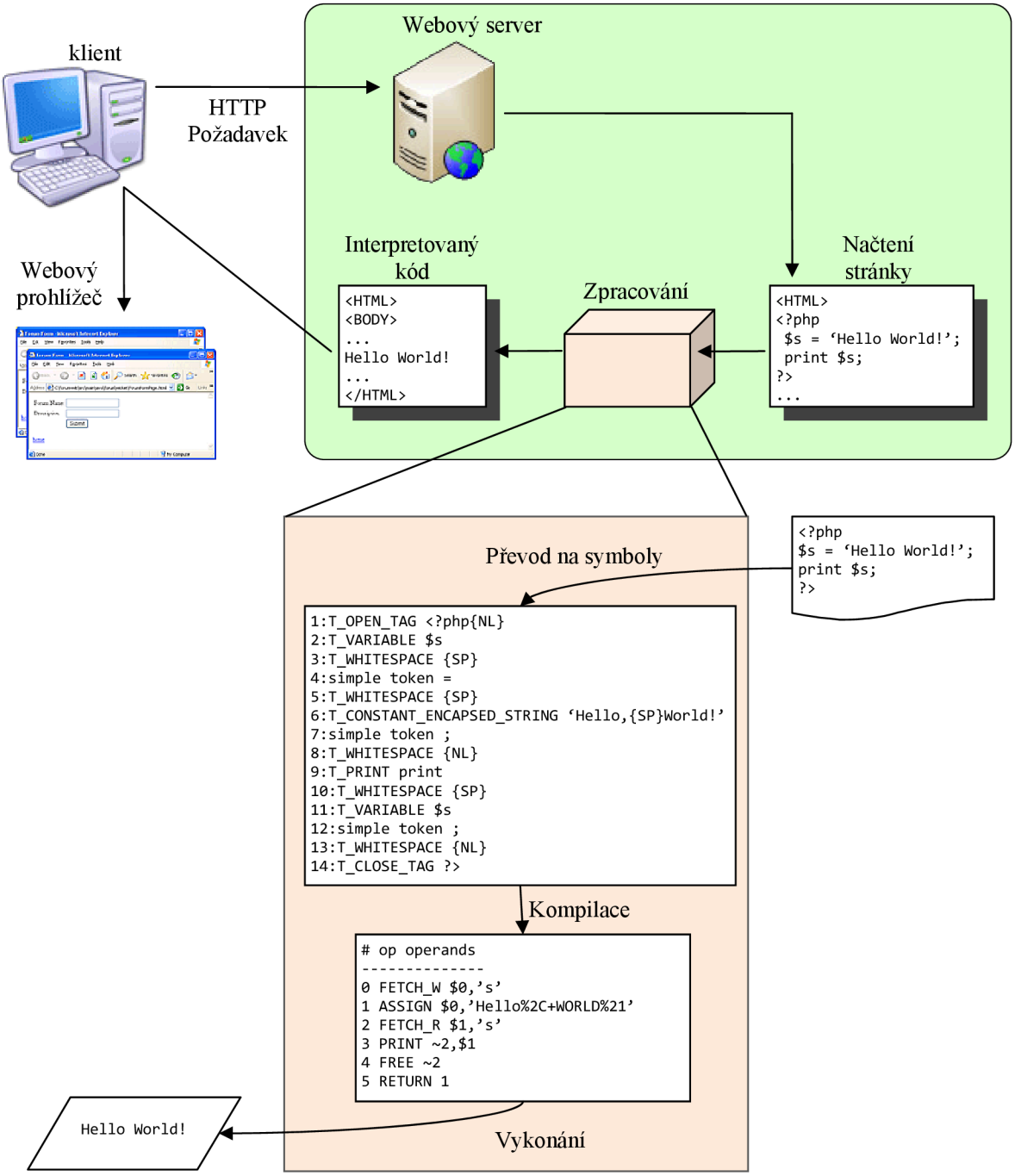
3.1.1 Jádro PHP

Prvek, který formuje jádro PHP, se nazývá Zend Engine. Stará se o strukturu jazyka – o to, že je tělo funkce uzavřené do složených závorek, že jsou na konci každého řádku středníky, a že klíčová slova jako `foreach`, `while`, nebo `else` mají speciální význam. Na hlavě Zend Engine sedí knihovny funkcí PHP, které dělají všechno možné, od generování dynamických obrázků až po zakódování speciálních znaků v URL.

Zend Engine stanovuje pravidla gramatiky, PHP poskytuje slovník pro programy. Engine definuje syntax jazyka, PHP definuje sémantiku. Ale skutečnost, že konkrétní funkce vykoná to, k čemu je určena, to je mimo obor zodpovědnosti Zend Engine. Engine tedy připraví sadu pravidel, jak se má definovat funkce a jak se má volat, ale už nic neříká o kódu, který konkrétní funkci implementuje. Následující podkapitola blíže popisuje funkci Zend Engine.

3.1.2 Průběh zpracování skriptu

Mezi okamžikem, kdy webový server obdrží požadavek ze stránky PHP, a okamžikem, kdy vrátí vygenerovaný výstup zpět klientovi, probíhá horečně velké množství činností. Obrázek Obrázek 3-1 znázorňuje činnosti při zpracování skriptu doplněné celým procesem od požadavku klienta na zobrazení stránky až po její výstup do prohlížeče.



Obrázek 3-1: Průběh zpracování skriptu od požadavku klienta po výstup do prohlížeče.

Hlavní tři kroky zpracování skriptu, znázorněné rovněž na obrázku Obrázek 3-1, jsou:

- **Převod na symboly** – Standardizuje potenciálně zmatečný text souboru PHP na série normalizovaných symbolů (anglicky *tokens*, celá činnost potom *tokenize*), které se budou Zend Engine snadněji zpracovávat.
- **Kompilace** – Převede seznam symbolů na instrukce pro Zend Engine, které jsou nazývány operační kódy (anglicky zkráceně *opcodes*). Každý operační kód informuje Zend Engine,

jakou konkrétní akci má podniknout, například sečíst dvě čísla, nebo vytisknout nějaký řetězec.

- **Vykonání** – Zend Engine postupuje podle instrukcí operačních kódů a vyprodukuje výsledky, jaké se od skriptu očekávají, například zobrazení textu nebo přístup k databázi.

Převod na symboly

Na počátku přečte Zend Engine soubor z disku a převede ho na symboly. Převod na symboly je proces, kdy se prohlédne celý soubor a převede se na série symbolů jazyka. V průběhu tohoto procesu se zjistí případné syntaktické chyby, a jakmile se na chybu narazí, proces skončí. Pokud program obsahuje více syntaktických chyb, vytiskne PHP chybovou zprávu a skončí poté, co narazí na první z nich. Nemůže-li parser převést celý program, nestará se o řízení další fáze procesu.

Každému symbolu přiděluje Zend Engine identifikační číslo. Například `$s` je symbol `T_VARIABLE` a identifikační číslo symbolu `T_VARIABLE` je 306. Zend Engine interně pracuje se symboly jen podle jim přiřazených čísel. Textová reprezentace, jako je `T_VARIABLE` slouží jen pro pohodlí lidí. Při sestavování seznamu symbolů se bere v úvahu každý znak v kódu včetně prázdných znaků (`T_WHITESPACE`).

Kompilace

Poté, co se zdrojový kód úspěšně převedl na symboly, začne pracovat kompilátor a převede posloupnosti symbolů na operační kódy. Podobně jako symboly formalizují text ze zdrojového kódu, formalizují operační kódy symboly na instrukce. Operační kódy jsou instrukce pro virtuální stroj PHP. Každý operační kód PHP se zpracovává pomocí nějaké funkce napsané v jazyce C uvnitř virtuálního stroje PHP.

Každá funkce a metoda třídy v programu PHP se kompiluje do seznamu operačních kódů, a kód s globálním oborem v programu se také kompiluje do seznamu operačních kódů. Kompilátor také vytvoří tabulky všech funkcí a tříd, které jsou v programu definované. Každý operační kód přebírá jako argumenty jeden nebo dva operandy a vrací výsledek.

Vykonání

Poté, co kompilátor vybuduje seznamy operačních kódů, předá se řízení tzv. *exekutorovi*, který začne pracovat na začátku seznamu operačních kódů, kde je seznam pro kód s globálním oborem, a začne jednotlivé operační kódy vykonávat. Jednotlivé operační kódy spouští exekutor tak, že zavolá jejich odpovídající funkce v C. Těmto funkcím v C se říká *zpracovatelé operačních kódů*. Např. zpracovatel operačního kód `MOD` provádí operaci zbytek po dělení specifikovanou svými operandy. Zpracovatel operačního kódu `PRINT` zavolá jinou funkci Zend Engine, která zase zavolá nějakou tiskovou funkci. Ve virtuálním stroji Zend Engine existuje okolo 150 operačních kódů, a každý z nich má svého vlastního zpracovatele.

Bez ohledu na to, o jak komplikovaný program se jedná, rozebere se každý program PHP na sadu seznamů operačních kódů, které se spouštějí ve virtuálním stroji Zend Engine. Proces převodu na symboly, kompilace a exekuce se týká hlavního skriptu PHP, i všech vkládaných skriptů. Převod na symboly a kompilace vkládaného souboru se děje až při běhu, když exekutor narazí na příkaz, jímž se soubor vkládá (`INCLUDE_OR_EVAL`). Z toho plyne, že všechny chyby při rozkladu,

nebo jiné závady ve vkládaných souborech, se neodhalí do té doby, dokud nezačne běžet hlavní skript.

3.2 Typy dat a proměnné v PHP

Jelikož je jedním z důležitých požadavků na nástroj této práce zobrazení aktuálních hodnot proměnných vyvíjené aplikace, věnuje se tato kapitola možným typům proměnných resp. typům dat uložených v proměnných a jejich charakteristikám v jazyce PHP.

Datový typ je obecný název přiřazený nějaké množině dat, která sdílí nějakou společnou množinu charakteristik. Datové typy v PHP lze rozdělit do tří kategorií: *skalární*, *složené* a *speciální* [13].

3.2.1 Skalární datové typy

Skalární datové typy mohou obsahovat jen jediný prvek informace. Do této kategorie spadá několik datových typů: `boolean`, `integer`, `float` a `string`.

Boolean – Booleovská proměnná reprezentuje pravdivost, a podporuje tedy pouze dvě hodnoty: TRUE (pravda) nebo FALSE (nepravda). Alternativou je reprezentace FALSE hodnotou nula, a TRUE jakoukoli nenulovou hodnotou.

Integer – Typ `integer` vyjadřuje celé číslo, neboli takové číslo, které nemá desetinnou část. Hodnotu lze vyjádřit dekadicky, oktalově nebo hexadecimálně.

Float – Tento typ vyjadřuje čísla s pohyblivou řádovou čárkou, tedy umožňuje specifikovat i desetinnou část čísel

String – Zjednodušeně řečeno je `string` (řetězec) nějaká posloupnost znaků, se kterou se zachází jako s jednolitou skupinou. Lze však přistupovat i k jednotlivým znakům řetězce.

3.2.2 Složené datové typy

Složené datové typy umožňují shromáždit do jediné entity více prvků téhož typu. Do této kategorie patří *pole* a *objekt*.

Pole – Pole (`array`) shromažďuje a uspořádává podobné prvky a odkazuje se na ně jistým specifickým způsobem. Formálně je pole indexovaná kolekce hodnot dat. Každý člen pole indexů se odkazuje na odpovídající hodnotu. Hodnoty pole mohou být opět datového typu pole, v takovém případě se jedná o *vícerozměrné pole*.

Objekt – Jedná se o datový typ, který je centrálním pojmem paradigmatu objektově orientovaného programování. Na rozdíl od ostatních datových typů PHP, musí být objekt deklarován explicitně. Deklarace charakteristik a chování objektu se děje uvnitř *třídy*, která zpravidla obsahuje několik atributů a funkcí, které objekt definují a popisují.

3.2.3 Speciální datové typy

Speciální datové typy zahrnují takové typy, které slouží k nějakému konkrétnímu speciálnímu účelu, a nejde je proto zařadit do žádné z ostatních kategorií typů. Patří sem `ressource` (prostředek) a neplatná, neznámá, či chybějící hodnota, neboli `null`.

Resource – PHP se často používá k interakci s nějakým externím zdrojem dat, jako např. databáze, soubory nebo proudy. Typicky se interakce uskutečňuje prostřednictvím *handle* – systémového identifikátoru prostředku, které se pojmenovává v době, kdy bylo úspěšně iniciováno připojení k tomuto prostředku. Dané handle zůstává ústředním odkazovacím bodem pro tento prostředek po celou dobu komunikace, a když komunikace skončí, odpovídající handle se zničí. Handle mají datový typ `resource`. Proměnné typu `resource` neobsahují skutečnou hodnotu; obsahují ukazatel na otevřené připojení prostředku.

Null – `Null` znamená, že hodnota není definována, nebo také že chybí. V PHP se hodnota považuje za `Null`, jestliže:

- Nebyla nastavena na žádnou předdefinovanou hodnotu.
- Byla jí konkrétně přiřazena hodnota `Null`.
- Byla odstraněna pomocí funkce `unset()`.

Datový typ `null` rozpoznává jen jedinou hodnotu, kterou je `Null`.

3.2.4 Superglobální proměnné

Kromě standardních proměnných nabízí PHP řadu užitečných předdefinovaných proměnných, které jsou přístupné z jakéhokoliv místa právě vykonávaného skriptu. Tyto speciální proměnné se nazývají *superglobální*. Poskytují vývojářům značný objem informací vztahujících se k prostředí. Superglobální proměnné resp. zobrazení jejich hodnot jsou také jedním z důležitých a očekávaných požadavků na nástroj této práce.

Superglobální proměnné v PHP tvoří devět předdefinovaných polí proměnných:

$_SERVER je pole obsahující informace poskytované hostitelským webovým serverem. Nabízí množství informací o konfiguraci serveru a klienta, a také o aktuálním prostředí pro požadavky.

$_GET obsahuje hodnoty, které se týkají parametrů předávaných metodou `GET`. Superglobální pole `$_GET` je standardně jediný způsob, jak lze přistupovat k proměnným předávaným metodou `GET`.

$_POST obsahuje hodnoty, které se týkají parametrů předávaných metodou `POST`. Superglobální pole `$_POST` je standardně jediný způsob, jak lze přistupovat k proměnným předávaným metodou `POST`.

$_COOKIE je pole, do kterého se ukládaní informace předávané do skriptu prostřednictvím HTTP cookies. Takové cookies typicky nastavil předchozí vykonávaný skript.

$_FILES obsahuje informace o datech, která byla na server nahrána metodou `POST`. Toto superglobální pole se od ostatních poněkud liší v tom, že je dvourozměrné s pěti prvky, obsahujícími informace o nahrávaném souboru.

$_ENV nabízí informace týkající se podkladového serverového prostředí parseru PHP, jako je např. hostitelský název serveru nebo systémový shell.

$_REQUEST zaznamenává proměnné předávané do skriptu přes jakoukoli vstupní metodu, konkrétně `GET`, `POST` a `Cookie`.

$_SESSION informace týkající se všech proměnných sezení (session) [4].

$GLOBALS lze chápat jako superglobální supermnožinu, protože obsahuje výpis všech proměnných s globálním oborem.

3.2.5 Konstanty

Stejně jako superglobální proměnné jsou konstanty přístupné z kteréhokoliv místa právě vykonávaného skriptu. Hodnota konstanty je po prvotní definici neměnná, nelze ji tedy modifikovat v průběhu vykonávání programu. Konstanty se definují funkcí `define()` a je běžné, že se názvy konstant píší výhradně velkými písmeny.

3.3 Chyby programu v PHP

Z hlediska ladícího nástroje této práce jsou chyby v programu nevyhnutelnou součástí práce. Začněme druhy chyb v programech.

3.3.1 Druhy chyb v programech

Ačkoli je PHP relativně dobré prostředí pro ladění a zpracování chyb, které umožňuje detekovat chyby a reagovat na ně, neexistuje jediná odpověď na všechny potřeby zpracování chyb, žádné proměnné nebo funkce pro vyhledávání všech chyb. Chyby mají mnoho příčin a lze se setkat s mnoha různými druhy chyb.

Chyby lze dělit podle různých kritérií. Podle [10] a [11] se v PHP a obecně v každém programovacím jazyce chyby v programech dělí na tři skupiny:

- *Syntaktické chyby* nebo také *chyby při kompilaci* – Tyto chyby nastávají při chybné syntaxi jazyka, tedy pokud kód programu není zapsán podle pravidel jazyka. Syntaktické chyby nastávají při analýze kódu ještě před jeho prováděním. V takovém případě parser PHP není schopen skript či jeho část zpracovat. Chyby syntaxe jsou nejlépe odhalitelné druhy chyb a PHP o nich vždy informuje chybovou zprávou s udáním čísla řádku, na kterém chyba nastala.
- *Sémantické chyby* – Sémantické chyby patří do kategorie *chyb při běhu programu*. Nastanou při provádění kódu, který je syntakticky správný. Nejsou detekovány

[4] Session představuje množinu proměnných, které dovolují uchovávat hodnoty, které jim byly nastaveny, po dobu připojení (tj. se znovunačtením stránky se neztratí).

analyzátorem PHP, ale nastanou, až když se má chybný řádek provést. Tyto chyby se eliminují hůře než chyby syntaktické, vznikají jako reakce na určitou kombinaci událostí, proto je obecně těžší je detekovat a opravit, než chyby syntaktické. Na sémantické chyby PHP reaguje rovněž udáním čísla řádku, na kterém chyba vznikla, ovšem původ chyby nemusí být na první pohled zřejmý.

- *Logické chyby* – Logické chyby jsou často jedny z nejhůře odhalitelných. Týkají se kódu, který je syntakticky i sémanticky v pořádku, ale nedělá to, co autor zamýšlel. Je obtížné je najít, protože nezpůsobují žádné chybové zprávy a nedají se detekovat automaticky. Právě tyto chyby by měl pomoci odhalit nástroj této práce.

Toto dělení chyb je podmnožinou jiného dělení [12], které uvádí dva základní druhy chyb:

- *Vnější chyby (externí)* nebo také *chyby prostředí* – jsou to chyby, které nejsou způsobeny chybou programátora, ale jsou výsledkem práce prostředí. Například spojení na databázi se nezdaří, když je to právě požadováno. Vnější chyby se mohou vyskytovat vždy, bez ohledu na to, jak je kód „bezchybný“, protože při jejich vzniku nezáleží na programu samotném, ale jeho okolním prostředí.
- *Chyby v kódu* – tyto chyby jsou přímo součástí kódu a patří mezi ně chyby syntaktické, sémantické a logické.

Typy chybových zpráv a úrovně chyb v PHP jsou popsány v následující podkapitole 3.3.2.

3.3.2 Úrovně chyb v PHP

Každá chyba, která v PHP aplikaci vznikne, je kategorizována podle své závažnosti. Mechanismus ošetření chyb poté implicitně vypíše chybovou zprávu společně s úrovní chyby, názvem zdrojového souboru a číslem problémového řádku. Úroveň chyby tedy oznamuje, k jak závažné chybě došlo. Zatímco méně závažné chyby nepřerušují běh aplikace, z těch závažnějších se aplikace již zotavit nemůže a způsobí přerušení jejího chodu.

Jednotlivé úrovně chyb se s postupným vývojem jazyka PHP neustále mění resp. přibývají, ty základní však zůstávají nezměněny. V tabulce Tabulka 3.1 jsou zobrazeny úrovně chyb aktuální verze jazyka PHP, tj. verze 5.3.0.

Hodnota	Název	Popis	Poznámka
1	`E_ERROR`	Kritické chyby při běhu	
2	`E_WARNING`	Varování při běhu	
4	`E_PARSE`	Chyby parseru PHP v době kompilace	
8	`E_NOTICE`	Oznámení – nezávažné chyby při běhu	
16	`E_CORE_ERROR`	Kritické chyby, k nimž dojde při startu PHP	od PHP 4
32	`E_CORE_WARNING`	Varování, k nimž dojde při startu PHP	od PHP 4
64	`E_COMPILE_ERROR`	Kritické chyby v době kompilace	od PHP 4
128	`E_COMPILE_WARNING`	Varování v době kompilace	od PHP 4
256	`E_USER_ERROR`	Chyby generované uživatelem	od PHP 4
512	`E_USER_WARNING`	Varování generované uživatelem	od PHP 4

1024	**E_USER_NOTICE**	Oznámení generované uživatelem	od PHP 4
2048	**E_STRICT**	Doporučení pro přenositelnost mezi verzemi PHP	od PHP 5
4096	**E_RECOVERABLE_ERROR**	Uživatelsky ošetřitelné závažné chyby při běhu	od PHP 5.2.0
8192	**E_DEPRECATED**	Upozornění na prvky a funkce, které přestanou být v budoucích verzích podporovány	od PHP 5.3.0
16384	**E_USER_DEPRECATED**	`E_DEPRECATED` generované uživatelem	od PHP 5.3.0
30719	**E_ALL**	Všechny chyby a upozornění	6143 v PHP 5.2.x, 2047 předchozí verze

Tabulka 3.1: Konstanty úrovní zpráv při chybě

Nejméně závažné jsou oznámení, naopak nejzávažnější jsou kritické chyby. Z číselných hodnot uvedených konstant se sestavuje jednočíselná hodnota (bitová maska), která jednoznačně udává typ hlášení úrovní chybových zpráv. Pokud se nastavení provádí v konfiguračním souboru `php.ini`, což je standardní postup, používají se názvy konstant spolu s logickými operátory AND, OR a NOT, které jsou zastoupeny znaky „&“„|“ a „~“, přičemž hodnota nastavení se ukládá do proměnné `error_reporting`. Výchozím nastavením PHP je generování chybové zprávy při všech chybách, kromě oznámení. Toto nastavení je zapsáno takto:

```
error_reporting = E_ALL & ~E_NOTICE
```

Mezi 4 základní úrovně chyb v PHP patří `E_PARSE`, `E_ERROR`, `E_WARNING` a `E_NOTICE`:

`E_PARSE`, Chyby parseru – Chyby při analýze jsou syntaktické chyby, které nastanou, porušuje-li kód programu pravidla jazyka PHP. Tyto chyby jsou detekovány parserem, který kontroluje syntaxi kódu před jeho provedením. PHP kompiluje skript do přechodného tvaru ještě před tím, než je proveden. Pokud kód obsahuje syntaktické chyby, skript nebude zkompilován.

`E_ERROR`, Kritické chyby – Kritické chyby jsou sémantické chyby nebo chyby prostředí, ze kterých se PHP nedokáže zotavit a způsobí okamžité zastavení provádění skriptu. To je způsobeno tím, že PHP nemůže pokračovat bez správného provedení příkazu, ve kterém nastala chyba. Příkladem takových chyb je volání nedefinované funkce nebo chyba při otevírání souboru v případě příkazu `require`.

`E_WARNING`, Varování – Pokud dojde k varování, pokusí se PHP pokračovat v provádění skriptu. Varování nastane, např. když příkaz `include` nemůže najít soubor, který má být vložen. Varování jsou obvykle chyby, které jsou natolik závažné, že by mohly zabránit správnému provedení programu. Mnoho chyb prostředí se projeví právě varovnými zprávami. Pokud v programu není žádné zpracování těchto chyb prostředí, způsobí to většinou výskyt dalších chyb v průběhu dalšího provádění skriptu.

`E_NOTICE`, Oznámení – Oznámení se zobrazují u méně závažných problémů, jako jsou např. neinicializované proměnné. Provádění skriptu většinou pokračuje bez problémů, ovšem oznámení mohou mít za následek logické chyby, které se špatně odhalují. Implicitně se totiž upozornění nevypisují.

3.3.3 Zpracování chyb v PHP

Nyní, když už je známo, jaké druhy chyb může PHP generovat, je vhodné uvést, jak s nimi lze zacházet, když nastanou. PHP poskytuje čtyři volby pro zpracování chyb, které spadají pod `error_reporting` [12]:

- Zobrazit je.
- Zapsat je do logu.
- Ignorovat je.
- Zachovat se podle nich.

Žádná z těchto voleb nenahrazuje jinou, co se týče důležitosti nebo funkčnosti; každá z voleb má své důležité místo v robustním systému pro zpracování chyb. Zobrazování chyb je velmi užitečné při vývoji, logování je zase obvykle vhodnější při ostrém běhu aplikace. Některé chyby lze bez obav ignorovat a některé vyžadují reakci. Přesnou směs technik pro zpracování chyb si ovšem každý volí sám podle svých potřeb.

Zobrazování chyb

Pokud je jako způsob zpracování chyb zvolena možnost chyby zobrazovat, jsou chyby posílány na standardní výstup, což v případě webových aplikací znamená, že jsou posílány ke zobrazení v prohlížeči. Tuto volbu lze přepínat pomocí nastavení v konfiguračním souboru `php.ini`:

```
display_errors = On
```

Takové nastavení je vhodné při vývoji aplikace. Naopak nevhodné je zobrazovat chyby koncovému uživateli při ostrém běhu.

Logování chyb

PHP vnitřně podporuje jak logování chyb do souboru, tak i logování do `syslog`, což je protokol operačního systému, do kterého se zaznamenávají různé události. `Syslog` a jeho mechanismus je pro každý druh operačního systému specifický. Nastavení logování chyb se provádí v konfiguračním souboru php.ini takto:

```
log_errors = on
```

a následující nastavení určuje logování do souboru:

```
error_log = /cesta/k/nazevsouboru
```

nebo do `syslog`:

```
error_log = syslog.
```

Logování poskytuje jasný přehled o všech chybách, které se ve webové aplikaci objeví. Kromě chyb, které zapisuje systém, lze chybovou zprávu v logu vytvořit i manuálně pomocí PHP funkce `error_log()`. Alternativně lze pomocí error_log() poslat zprávu na zadanou emailovou adresu nebo zapsat do zadaného souboru.

Ignorování chyb

PHP podporuje selektivní potlačení informací o chybách, které by mohly nastat, pomocí syntaxe `@`. Např. pokud by bylo cílem otevřít soubor, který nemusí existovat a v důsledku toho potlačit chyby, které tím pádem mohou nastat, lze použít následující:

```
$fp = @fopen($file, $mode);
```

Protože systém chyb v PHP neposkytuje žádné možnosti řízení toku programu, je možno potlačit pouze chyby, o kterých je známo, že mohou nastat.

Reagování na chyby

PHP umožňuje pomocí funkce `set_error_handler()` nastavení uživatelsky definovaných funkcí pro zpracování chyb. Pokud je taková funkce definována a PHP detekuje chybu, namísto toho, aby byla zobrazena nebo zapsána do chybového logu, bude zpracována definovanou funkcí. V případě méně závažné chyby, pokud je její zpracování kompletní, pokračuje skript od místa, kde nastala chyba; v případě fatální chyby skript po zpracování chyby končí.

3.4 Dokumentace

[12] Dokumentace je důležitou součástí pro vytváření kvalitního kódu. Ačkoli je správně napsaný kód schopen samopopisnosti, programátor musí neustále kód číst, aby pochopil jeho funkčnost.

Dokumentaci k nástroji této práce můžeme rozdělit do dvou hlavních kategorií:

- Vnořené komentáře, které jsou vepsány přímo do kódu a které mají hlavně za úkol zjednodušit pochopení, opravy nebo ladění kódu.
- Dokumentace API (z angl. Application Programming Interface – aplikační programátorské rozhraní) pro uživatele, kteří chtějí dané třídy nebo funkce používat, bez toho, aby byli nuceni pročítat kód jako takový.

K první kategorii lze dodat, že vnořené komentáře by měly sloužit k objasnění kódu, nikoli aby pouze popisovali to, co jednotlivý řádek přímo provádí (např. komentář jako „inkrementace proměnné i“). Nepotřebné komentáře pouze znepřehledňují kód. Vnořené komentáře by měly také popisovat to, co nemusí být na první pohled patrné.

3.4.1 Dokumentace API

Dokumentace API je určena pro externí uživatele a je odlišná od dokumentace psané přímo do kódu. Cílem API dokumentace je to, aby programátor mohl výsledný nástroj používat, aniž by se musel podívat do kódu.

Základní požadavky na API dokumentaci jsou:

- Měla by poskytovat základní popis nástroje tak, aby se koncový uživatel mohl rychle rozhodnout, zdali je nástroj vhodný pro řešení jeho problémů.
- Měla by obsahovat kompletní seznam všech veřejných tříd a funkcí a jasný a přehledný popis jejich vstupních a výstupních parametrů.

A dále je často vhodné poskytnout koncovému uživateli:

- Návody a příklady použití kódu, které názorně ukazují, jak může být daný kód použit.
- Dokumentaci chráněných metod.

Systém tvorby API dokumentace by měl s ohledem na programátory, kteří píší kód, který má být později zdokumentován, splňovat následující rysy:

- Text dokumentace by měl být umístěn přímo v kódu. To je užitečné proto, aby dokumentace byla aktuální, a také to podává informaci o tom, že daná dokumentace existuje.
- Dokumentační systém by měl mít jednoduchou a účelnou syntaxi.
- Měl by být použit systém pro generování „zkrášlené“ dokumentace. To znamená, že dokumentace by měla být schopna jednoduchého převodu do profesionálního a jednoduše čitelného formátu.

K vytvoření API dokumentace lze využít již existujících balíků, které automaticky vytvoří API dokumentaci ze správně strukturovaných vnořených komentářů. Takovými balíky jsou například *Doxygen* nebo pro PHP specifičtější *phpDocumentor*.

3.5 Specifikace funkcionálních požadavků

Po bližším zkoumání požadavků na funkce nástroje uvedených výše v podkapitole **Chyba! Nenalezen zdroj odkazů.** lze některé požadavky dále rozšířit a zkonkretizovat.

Zobrazení aktuálních hodnot proměnných

Funkci zobrazení aktuálních hodnot proměnných lze konkrétněji rozdělit na několik samostatných funkcí:

a) **Zobrazení hodnot superglobálních proměnných** – Jedná se o zobrazení obsahu všech superglobálních proměnných, které se týkají laděné aplikace nezávisle na uživatelské iniciativě.

b) **Zobrazení definovaných konstant** – Konstanty sice nepatří mezi superglobální proměnné, ale jsou rovněž přístupné z každého bodu zdrojového kódu, proto lze kromě superglobálních proměnných zobrazovat i tyto konstanty.

c) **Zobrazení aktuálních hodnot uživatelem vybrané proměnné** – Tato funkce zobrazuje konkrétní proměnnou aplikace, kterou uživatel cíleně označil k výpisu její/jejich hodnoty/hodnot.

Doba vykonání skriptu

Funkce zobrazení doby, za kterou se vykoná kód skriptu, lze z hlediska požadavků rozlišit na dva typy měření doby:

a) **Celková doba vykonání celého skriptu** – jedná se o funkci zobrazení celkové doby vykonávání aktuálního skriptu.

b) **Doba vykonání skriptu mezi trasovacími body** – jedná se o funkci zobrazení doby vykonávání aktuálního skriptu mezi jednotlivými uživatelem zadanými body ve skriptu

Oznámení a informace o chybách

Systém zpracování chyb poskytuje několik funkcí:

a) **Oznámení, kde ve zdrojovém kódu je chyba** – pokud nastane chyba v aplikaci, nástroj by měl zobrazit, kde tato chyba nastala, na jakém řádku a v jakém souboru. Také je možno zobrazit část zdrojového kódu s chybou.

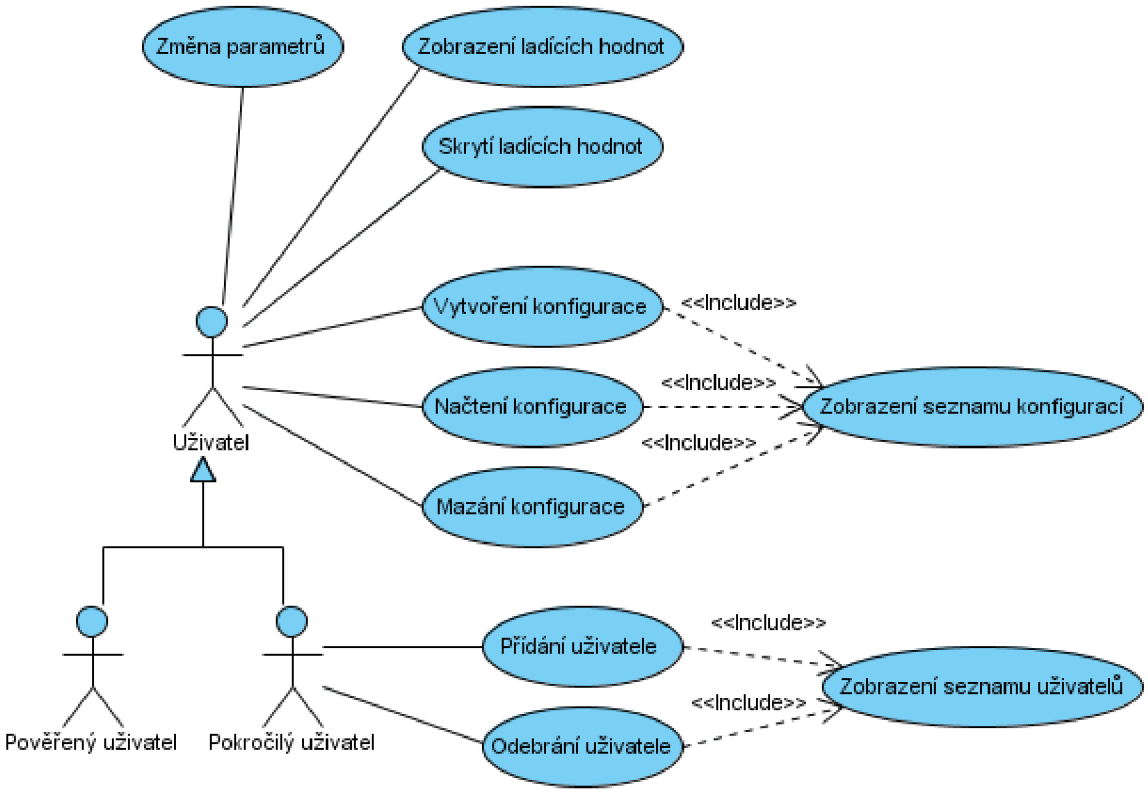
b) **Zobrazení informací o chybách** – pokud nastane chyba v aplikaci, nástroj by měl zobrazit typ chyby, popis chyby a chybovou zprávu.

c) **Trasování chyb** – pokud nastane chyba v aplikaci, nástroj by měl vhodným způsobem zobrazit cestu zdrojovým kódem, přes kterou byl vykonáván běh programu, než chyba nastala.

3.6 Diagram případů použití

Diagram případů použití se vytváří pro znázornění funkcionálních požadavků na systém. Diagram představuje soubor případů použití, aktérů a jejich vzájemných vztahů. Každý případ použití popisuje posloupnost událostí. Každá posloupnost je inicializována jistou entitou, která se nazývá *aktér*.

Zjednodušený diagram případů použití nástroje pro usnadnění vývoje a testování PHP aplikací této práce je znázorněn na obrázku Obrázek 3-2. Jelikož je funkcionálních požadavků pro zobrazení v diagramu mnoho, byly prvky s podobnou funkcionalitou sloučeny do obecných případů použití s názvy „Zobrazení ladících hodnot“ a „Skrytí ladících hodnot“.



Obrázek 3-2: Zjednodušený diagram případů použití

Specifikace např. případu použití *Přidání uživatele* znázorněného v diagramu je následující:

Případ použití: Přidání uživatele
ID: PP7
Stručný popis: Přidání nového uživatele do systému ladícího nástroje.
Účastníci: Pokročilý uživatel
Vstupní podmínky: Autentizovaný *Pokročilý uživatel*.
Hlavní tok: 1. Systém nejprve zobrazí přehled všech dostupných uživatelů systému spolu se vstupními poli pro přidání nového uživatele. 2. Případ použití se spustí, když *Pokročilý uživatel* zvolí „Přidat nového uživatele“. 3. Pokud nejsou všechny povinné údaje vyplněny, systém vyžaduje, aby *Pokročilý uživatel* zadal potřebné údaje. 4. Při správně vyplněných údajích systém uloží nového uživatele. 5. Systém zobrazí nového uživatele v přehledu dostupných uživatelů.
Následné podmínky: Žádné.
Alternativní toky: NeplatnéÚdajeUživatele

Alternativní tok: Přidání uživatele: NeplatnéÚdajeUživatele
ID: PP7.AL1
Stručný popis: Systém zruší přidávání nového uživatele s návratem k zadávání údajů
Účastníci: Pokročilý uživatel
Vstupní podmínky: Autentizovaný *Pokročilý uživatel*.
Hlavní tok: 1. Alternativní tok se spustí po kroku PP7.3 2. Systém zruší přidávání uživatele do systému 3. Systém se vrací do kroku PP7.2
Následné podmínky: Žádné.
Alternativní toky: Žádné.

Ostatní specifikace případů použití znázorněných v diagramu na obrázku Obrázek 3-2 jsou uvedeny v *Příloha A* této práce.

Kapitola 4

Existující nástroje

Existující nástroje, které mají podobnou funkčnost jako nástroj této práce, lze rozdělit do tří základních kategorií, podle způsobu jejich použití:

a) Integrovaná vývojová prostředí,

b) Moduly a rozšíření PHP pro ladění,

c) Třídy a funkce pro ladění.

4.1 Integrovaná vývojová prostředí

Mezi většinou vývojářů napříč programovacími jazyky získala velkou oblibu integrovaná vývojová prostředí (IDE - Integrated Development Environment), jejichž funkčnost je popsána v podkapitole 2.1.3 této práce. Spousta z nich, zejména těch lepších, jsou však komerční, dostupné pouze za poplatek. Jako nekomerční integrovaná vývojová prostředí, které jsou dostupné zdarma a používají ladící nástroj, uvedu například tyto:

- Pro operační systém Windows je to **Maguma Studio Free**, které jako jedno z mála bezplatných vývojových prostředí pro Windows obsahuje vestavěný ladící nástroj, a to díky integrovanému PHP interpretu.
- Pro operační systém Linux je to prostředí **Quanta Plus**, pomocí kterého lze ladit PHP aplikace za pomocí přídavného ladícího nástroje Gubed, popsaného níže v podkapitole 4.2.3.

Integrovaná vývojová prostředí, co se využití znalostí o nich týče, mají ze všech tří kategorií k nástroji této práce nejdále. Jelikož jsou algoritmy a postupy jejich ladícího nástroje pevně spojeny a ukryty v nich samotných, naskýtají se jako využitelné pouze takové znalosti, které popisují, co ladící nástroj dělá, nikoli jak to dělá. To jest např., co vše může ladící nástroj provádět či zobrazovat.

4.2 Moduly a rozšíření PHP pro ladění

Kromě mnoha vestavěných funkcí jazyka PHP je k dispozici také mnoho funkcí v rozšířeních. Tyto rozšíření se dají snadno přidávat k instalaci PHP na serveru. V případě operačního systému Linux se většinou rozšíření tzv. přikompilovávají k již nainstalovanému aplikačnímu serveru PHP, v případě operačního systému Windows se většinou k souborům aplikačního serveru PHP nahrávají soubory rozšíření a mění určité řádky konfiguračních souborů.

Poznatky o modulech a rozšířeních PHP pro ladění, vzhledem k nástroji práce, lze v převážné většině rovněž využít jen pro popis, co má ladící nástroj dělat, nikoliv jak. Není to ovšem způsobeno ukrytím jejich způsobu fungování, jako u IDE. Oproti IDE je zde situace o něco lepší. Moduly a

rozšíření, které vykonávají funkci ladícího nástroje, jsou ve většině případů šířeny bezplatně spolu se zdrojovými kódy a dokumentací. Nahlédnutím do dokumentace nebo zdrojových kódů lze tedy porozumět jejich způsobu fungování.

Následující podkapitoly uvádí několik takových rozšíření, pomocí kterých se dají internetové aplikace ladit.

4.2.1 Xdebug

Xdebug je rozšíření založené na jádře Zend. Toto rozšíření obsahuje spoustu ladicích funkcionalit, jako sledování volání funkcí, alokace paměti, profilování (tj. sledování doby vykonávání jednotlivých funkcí) atd.

[14] Instalace probíhá buď příkazem `pear install xdebug` nebo stáhnutím binárních modulů Xdebug a následném zkopírování do patřičného umístění. Dále je třeba nastavit informace v konfiguračních souborech aplikačního serveru PHP.

Funkce pro sledování

Jak bylo zmíněno, Xdebug dokáže sledovat volání funkcí v aplikacích. Pokud chce uživatel zobrazit seznam všech volání funkcí v aplikaci, nejprve je třeba zavolat funkci `xdebug_start_trace()`, čímž je sděleno Xdebug, aby začal sledovat volání funkcí. Pak je třeba zavolat `xdebug_dump_function_trace()`, aby se vytiskl seznam volání od chvíle, kdy bylo voláno `xdebug_start_trace()`. Výstup je potom tabulka HTML s jedním řádkem pro každé volání funkce aplikace. Tabulka ukazuje uplynulý čas, okamžik, kdy se daná funkce zavolala, úroveň vnoření atd. Xdebug také umí sledovat parametry volané funkce, pokud to uživatel nastaví a vše se dá zapisovat do souboru.

Profily funkcí

Profily funkcí v Xdebug poskytují zprávy o tom, kde aplikace tráví při běhu svůj čas. To umožňuje vyhledat ty části kódu, které jsou pomalé, a je proto třeba zdokonalit. Profil funkce uživateli prozradí nejen to, jak dlouho trvalo volání nějaké funkce, ale i kolikrát se funkce volala. Profilování funkcí se odstartuje zavoláním `xdebug_start_profiling()`. Tím se sdělí Xdebug, aby začal shromažďovat statistiky o aplikaci. Ke zobrazení informací o profilech funkcí slouží zavolání `xdebug_dump_function_profile()`. Zpráva o profilech obsahuje seznam všech volaných funkcí, kolik času aplikace strávila v jednotlivých funkcích, kolikrát se jednotlivé funkce volaly a zdrojový soubor a číslo řádku, odkud se funkce volaly. Informace jsou seřazené podle čísel řádků skriptu, což lze nastavením změnit. Rovněž informace z profilů se dají uložit do souboru.

4.2.2 DBG

DBG je část balíku z PHPEd, což je název integrovaného vývojového prostředí. Je šířen ve dvou verzích, přičemž verze 3.1.x je součástí zmiňovaného IDE a je za peníze, verze 2.15.x je distribuovaná samostatně a je zdarma. Instalace nekomerční verze ladícího nástroje spočívá v nakopírování souboru dbg.so (linux) nebo php_dbg.dll (Windows) do složky s php rozšířeními a v editaci souboru php.ini. Úspěšnost instalace lze zjistit pomocí skriptu phpinfo(), kde se objeví

informace o modulu dbg. Zatímco pod Windows existuje k DBG nástroj pro grafický výstup, pod Linuxem je zdarma k dispozici pouze řádkový klient dbg-cli, který však může být použit k ladění skriptů pod GNU debuggerem DDD.

4.2.3 GUBED

GUBED je debugger postavený na samotném PHP, díky čemuž se kvůli němu nemusí instalovat na server žádný modul PHP ani žádné rozšíření serverového software. Není to tedy typické rozšíření jako předešlá dvě. To mimo jiné znamená, že můžete lze ladit skripty ve skutečném prostředí serveru, na němž budou běžet. GUBED vyžaduje kromě serverové části i nějaký klientský software, který bude výsledky ladění odchytávat. Tím softwarem může být buďto editor Quanta, nebo s projektem dodávaný nástroj nazvaný wxGubed. Instalace serverové části GUBEDu je triviální. Stačí nakopírovat adresář Gubed z instalačního balíčku do kořenové složky webu a vytvořit soubor s konfigurací.

4.2.4 Advanced PHP Debugger

Advanced PHP Debugger (APD) [12] byl vytvořen jako rozšíření PHP. APD potlačuje volání Zend Engine, aby poskytoval co nejpřesnější měření času. Vytváří trasovací soubory, které jsou strojově čitelné, a mohou být mnoha různými způsoby dodatečně zpracovány. Současně umožňuje uživatelské nastavení výstupního formátování, což umožňuje zobrazit výsledky profilování v prohlížeči, ve formátu XML nebo použít libovolný jiný formát. Dále nabízí krokovací, interaktivní ladící program. APD je součástí knihovny PECL (PHP Extension Community Library) a lze jej tedy instalovat pomocí instalátory PEAR (PHP Extension and Application Repository).

APD zaznamenává při běhu skriptu události, jako jsou:

- Započetí provádění funkce.
- Ukončení provádění funkce.
- Požadavek na zahrnutí souboru nebo čtení souboru.

K těmto událostem měří i čas jejich vykonání.

4.3 Třídy a funkce pro ladění

Pokud se vývojáři jeho dílo vydaří, či se chce pochlubit, nebo ze spousty jiných důvodů, může toto dílo volně zpřístupnit ostatním. Nejsnadnější a nejpoužívanější cestou je zveřejnění díla na internetu jako open source, neboli otevřený zdrojový kód. Takové dílo je potom dostupné a volně k používání pod nějakou licencí určenou autorem. Nejčastějšími typy licence bývá

- povolení používání kódu, ale tento použitý kód v jiné aplikaci musí být rovněž přístupný každému,
- povolení používání kódu, ale vždy se musí spolu s kódem uvést autor popř. jiné informace, které autor požaduje,
- povolení používání kódu, ale pouze pro nekomerční účely,

- aj.

Jedna z nejznámějších internetových stránek zaměřující se na distribuci volně dostupných tříd napsaných v PHP, která může být nápomocná především, pokud je vyvíjen kód nebo aplikace, jež se má skládat ze tříd, je stránka s názvem PHP Classes Repository [15]. Na uvedené stránce je možno nalézt hned několik tříd, které by mohly být nápomocné k vytváření nástroje této práce. Jsou jimi např.

Adump – Zobrazí proměnné aplikace do aktuálního nebo vyskakujícího (angl. popup) okna. Třída umí pracovat i s objekty. Zobrazení se mohou ukládat do superglobální proměnné SESSION.

bib_errors – Tato třída zachycuje chyby PHP skriptu za jeho běhu. Tyto chyby jsou zapisovány do souboru, mohou být zobrazovány v aktuálním okně nebo mohou být poslány na zadaný email.

Convert Class To HTML Table – Třída je určena ke zobrazení struktury nějaké třídy HTML tabulkou. Zobrazovaná třída může mít proměnné, které obsahují objekty jiných tříd.

debug – Do skriptu se umístí příkazy pro ladění a je umožněno zastavit nebo pokračovat ve vykonávání skriptu.

debug_var – Zobrazí obsah proměnné v dobře čitelném formátu zavoláním jedné funkce.

Gvar – Převádí POST a GET proměnné na globální proměnné, když je direktiva `register_globals` nastavena na „off“.

MDS Debug – Tato třída umí zobrazit obsah proměnných a také zobrazuje čas trvání vykonávání skriptu. Čas trvání vykonávání skriptu může být určen i dvěma body ve skriptu.

Tato kategorie existujících nástrojů, tedy třídy a funkce pro ladění, je kategorie nejvíce podobná nástroji této práce. Poznatky a znalosti o této kategorii lze efektivně využít při tvorbě nástroje této práce.

Kapitola 5

Návrh řešení

Dobrý návrh je podmínkou úspěšné tvorby softwarového produktu a minimalizuje náklady spojené s implementací a posléze údržbou produktu. Tato kapitola se věnuje návrhové etapě vývoje nástroje, o němž práce pojednává. Věnuje se uživatelskému rozhraní, uvádí některé principy fungování částí nástroje a popisuje nástroj z různých pohledů za využití návrhových diagramů.

5.1 Uživatelské rozhraní

Než se dostaneme k návrhu uživatelského rozhraní, je třeba zmínit některé důležité poznatky o možnostech interakce uživatele s nástrojem.

5.1.1 Interakce uživatele s nástrojem

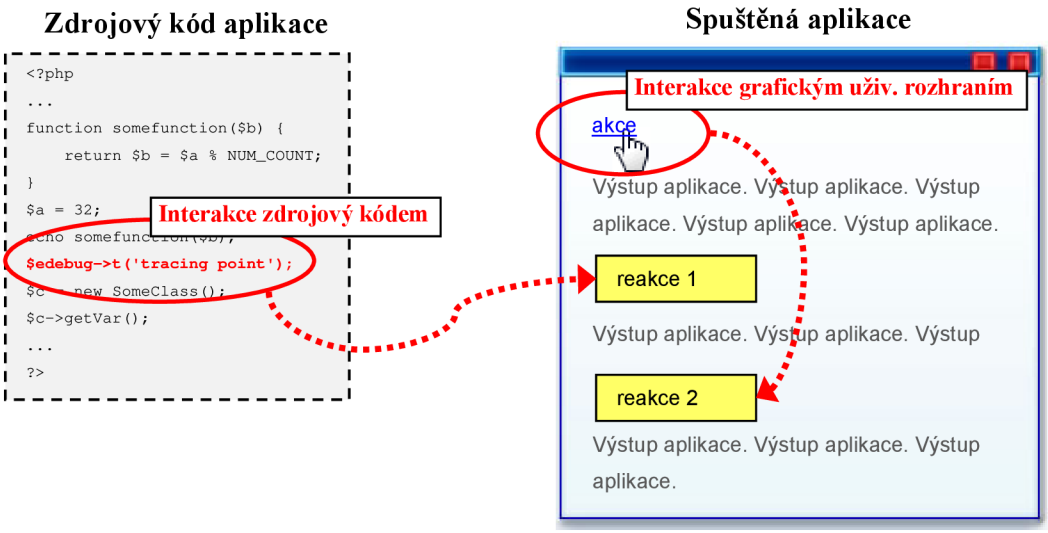
Způsob používání a interakce uživatele s nástrojem je oproti standardním aplikacím mírně odlišný. Zatímco většina běžných aplikací je ovládána přes grafický výstup - grafické uživatelské rozhraní aplikace pomocí uživatelských událostí, jako je kliknutí myší na vyobrazené tlačítko nebo stisk kláves na klávesnici, v případě nástroje této práce tvoří takový způsob interakce s aplikací jen jednu ze dvou možností.

Jelikož je praktické používání nástroje určeno výhradně vývojářům softwarových produktů (použití jiným typem uživatelů i mírně postrádá smysl) a vzhledem k typu a specializaci nástroje, je druhou možností interakce uživatele (tedy vývojáře softwarové aplikace) s nástrojem vkládání úseků kódu definovaných nástrojem přímo do zdrojového kódu vyvíjené aplikace. Tento způsob je nepřímou interakcí s nástrojem a projeví se až po spuštění aplikace.

Způsob používání nástroje lze tedy rozdělit na dva základní typy:

- **Interakce zdrojovým kódem** – vložení nástrojem definovaného úseku kódu do zdrojového kódu vyvíjené aplikace.
- **Interakce grafickým uživatelským rozhraním** – ovládání přes grafické uživatelské rozhraní nástroje.

Obrázek Obrázek 4-1 tyto dva typy interakce znázorňuje.



Obrázek 4-1: Znázornění typů interakce uživatele s nástrojem

Interakce zdrojovým kódem

Nástroj bude k tomuto účelu poskytovat tři základní funkce:

a) **d()** – Funkce pro zobrazení aktuální hodnoty a struktury uživatelem vybrané proměnné.

b) **t()** – Funkce pro vytváření uživatelem definovaných bodů ve zdrojovém kódu a vypsání informací o bodu, včetně uživatelova popisu bodu.

c) **tError()** – Funkce pro vytváření uživatelem definovaných chybových zpráv v určitém bodě zdrojového kódu a vypsání informací o zprávě.

Interakce grafickým uživatelským rozhraním

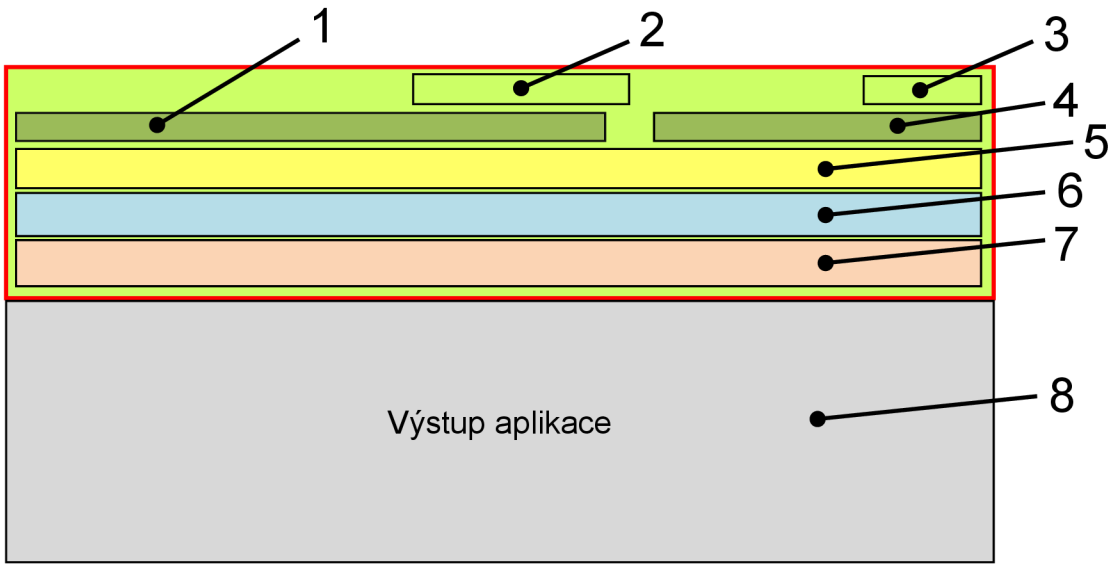
Jde o standardní ovládání nástroje přes jeho grafické uživatelské rozhraní pomocí standardních způsobů, jako jsou ovládání myší nebo vstup z klávesnice. Pomocí této interakce může uživatel provádět všechny ostatní úkony, které nevyplývají z uvedené interakce se zdrojovým kódem a plynou z požadavků na nástroj.

Podobu grafického uživatelského rozhraní popisuje následující podkapitola.

5.1.2 Grafické uživatelské rozhraní, layout

Layout znamená základní, zpravidla kreslený koncept obsahového a grafického rozvržení dokumentu (velikost, umístění, barvy, atd.). Při návrhu layoutu by měl být brán ohled na přehlednost a čitelnost dokumentu.

V případě nástroje této práce je žádoucí, aby se výstup nástroje nemíchal s výstupem laděné aplikace, pokud to uživatel explicitně nevyžaduje. Proto je vhodné umístit veškerý výstup nástroje do jedné oblasti. S ohledem na míru použitelnosti různých umístění se jako nejvhodnější jeví zobrazení výstupu nástroje nad výstupem aplikace. Návrh layoutu je prezentován následujícím obrázkem Obrázek 4-2.



Obrázek 4-2: Grafický layout nástroje, rozmístění komponent

Popis jednotlivých částí z obrázku Obrázek 4-2:

1 – Tato oblast je určena pro hlavní menu nástroje. Bude obsahovat položky pro výpis informací o uživateli, globálních proměnných, konstant a seznam souborů vložených do aplikace pomocí funkcí `include` a `require`. Pomocí položek tohoto menu se bude měnit obsah oblasti 5.

2 – Oblast s číslem 2 obsahuje titulek nástroje a jeho verzi.

3 – V této části bude položka pro skrytí či zobrazení celého obsahu nástroje pod položkou.

4 – Místo označené číslem 4 je vyhrazeno pro druhé menu. Bude obsahovat dobu vygenerování skriptu, odkaz na výpis serverových informací a nastavení a položku pro zobrazení nastavení nástroje.

5 – Oblast 5 slouží pro zobrazení dat vybraných ve hlavním menu v oblasti 1.

6 – V tomto místě se budou zobrazovat informace z trasovacích bodů definovaných uživatelem ve zdrojovém kódu aplikace.

7 – Oblast s číslem 7 zobrazuje uživatelem definované chyby a globální chyby aplikace.

8 – Osmá oblast je samotný výstup laděné aplikace a nepatří mezi komponenty nástroje.

Obsahové a grafické rozvržení nástroje bude zpracováno pomocí blokových elementů jazyka XHTML doplněno o kaskádové styly upravující vzhled.

5.2 Principy fungování některých prvků nástroje

5.2.1 Zabezpečení

Zabezpečení nástroje této práce se týkají především dva pojmy – *autentizace* a *autorizace*.

Autentizace je proces, při kterém se ověřuje, zda je uživatel nebo entita opravdu ten, za koho se vydává.

Autorizace je proces, při kterém se ověřuje, zda má uživatel dostatečná oprávnění pro přístup k určitému souboru či pro provedení určité akce. Tato kontrola se provádí na základě členství uživatele v různých uživatelských skupinách, přístupových seznamech apod. Předpokládá se předchozí úspěšná autentizace, na jejíž spolehlivosti je autorizace závislá.

Princip ověření totožnosti v případě nástroje této práce bude fungovat na základě zjištění IP adresy uživatele a porovnání této IP adresy na výskyt v seznamu přístupových adres uchovávaného nástrojem. U každé IP adresy bude rovněž uložena informace o příslušných uživatelských právech. Takto je splněna autentizace i autorizace a navíc odpadá nutnost se přihlašovat, což může být i nevýhodou, ovšem z hlediska tohoto konkrétního typu nástroje je tento bezpečnostní systém vyhovující.

5.2.2 Zobrazení aktuálních hodnot proměnných

Jedná se o princip funkce zobrazení hodnot proměnných různého typu včetně polí, objektů, superglobálních proměnných jako $_SESSION, $_GET, $_POST, $_COOKIE atd.

Nejprve uživatel v aplikaci resp. v kódu aplikace vybere proměnnou, kterou chce zobrazit. Dále zavolá v patřičné části kódu metodu pro výpis hodnoty s argumentem proměnné a popřípadě i argumentem názvu proměnné. Volaná metoda vytiskne hodnotu proměnné na výstup v úhledném formátu.

Metoda, která pracuje s hodnotou proměnné získané z parametru, nejprve zjistí typ proměnné a podle typu zpracuje výstupní informace. Pokud je zadán název proměnné, začlení ho do výstupních informací, v opačném případě nikoliv.

5.2.3 Trasování chodu skriptu

Jedná se o funkci zobrazení uživatelem zadaných bodů ve skriptu a vypsání informací o bodě, případně času. Princip provedení je podobný principu zobrazení aktuálních hodnot proměnných.

Uživatel nejprve v aplikaci resp. v kódu aplikace vybere bod, ve kterém chce provést trasování. Dále zavolá v patřičné části kódu metodu pro provedení trasování. Volaná metoda vytiskne informace o trasovacím bodu na výstup v úhledném formátu.

5.2.4 Ukládání konfigurace

Dle požadavků se má konfigurace nástroje ukládat jak lokálně, přístupná z konkrétního místa, odkud je nástroj používán, tak globálně, přístupná z kteréhokoliv místa používání nástroje.

Lokální uložení, což znamená uložení na straně klienta, lze v případě webových aplikací realizovat pomocí cookies, které se ukládají v prohlížeči. V případě lokálního uložení tedy nástroj uloží data do těchto cookies prohlížeče.

Globální uložení lze realizovat několika způsoby. Nejpřijatelnějšími z řešení jsou ukládání do databáze a ukládání do souboru umístěného na stejném serveru jako nástroj. Pokud by se konfigurace ukládaly do databáze, bylo by třeba zajistit další kroky k vytvoření spojení a ovládání databáze. Také by vznikly další povinnosti při instalování a zprovozňování nástroje. Toto vše odpadá, pokud se budou globální konfigurace ukládat do souboru. Řešení je tedy ukládáni do souboru.

Struktura dat v souboru pro uložení konfigurací by měla být vhodného formátu, kvůli snadnému zpracování a hlavně možnosti přenositelnosti. Z dostupných formátů se nabízí populární a hojně užívané XML.

XML

XML (eXtensible Markup Language) je obecný značkovací jazyk, který umožňuje vytváření širokého spektra různých typů dat a konkrétních značkovacích jazyků pro různé účely. Byl vyvinut a standardizován konsorciem W3C.

XML je otevřený formát, což znamená, že jeho specifikace je každému zdarma k dispozici na serveru konsorcia W3C. Každý tak může do svých aplikací bez problémů implementovat podporu XML. To představuje velký rozdíl oproti firemním formátům, k nimž není k dispozici žádná dokumentace a navíc se jedná v porovnání s XML o leckdy velmi složité, často binární formáty.

Práci s XML usnadňuje fakt, že je celý formát založen na prostém textu. I když pro většinu lidí zůstane kód XML skryt a budou ho používat pouze aplikace pro vzájemnou komunikaci, není problém kdykoliv otevřít XML dokument v textovém editoru a potřebné úpravy provést ručně.

Jako znakovou sadu používá ISO 10646, což je 32 bitová sada, která dokáže pojmout všechny znaky dnes používaných jazyků. Proto lze v XML míchat různé znaky různých jazyků dohromady. Pokud by ovšem dokumenty obsahovaly např. pouze český text, znamenalo by ukládání přímo v ISO 10646 zbytečné plýtvání místem. Proto lze v každém XML dokumentu určit specifické kódování.

Pomocí XML značek se vyznačuje v dokumentu význam jednotlivých částí textu. Dokumenty obsahují mnohem více informací, než kdyby se používalo prezentační značkování (např. „tohle je tučným písmem Arial o velikosti 12 bodů zarovnané vlevo“). XML dokumenty jsou informačně bohatší, což lze využít v mnoha oblastech. Velký přínos to znamená pro vyhledávání.

XML také umožňuje definovat vlastní sadu značek, která se bude v dokumentu používat a pomocí parseru kontrolovat, zda dokument obsahuje pouze povolené značky (čerpáno z [16]).

5.2.5 Inicializace a způsob zobrazení nástroje

Z hlediska použitelnosti a snadnosti instalace je vhodné, aby kód nástroje co nejméně zasahoval do kódu laděné aplikace. Proto kroky k jeho instalaci a používání musí být minimální.

Pro začlenění nástroje do aplikace se ve zdrojovém kódu aplikace pouze vloží řádka kódu na začátek a konec stránky, která má být laděna. Javascript potom pomocí modifikace struktury DOMu stránky zařídí vložení potřebných elementů k začlenění kaskádových stylů a javascriptu, které nástroj využívá. Pomocí javascriptu a DOMu se také provede zobrazení nástroje a jeho komponent na patřičných místech stránky.

5.3 Struktura

Tato podkapitola popisuje strukturu výsledného nástroje formou návrhových diagramů, které spadají do jazyka *UML*. Konkrétně podkapitola obsahuje diagramy struktury, které spadají do kategorie modelování statické struktury aplikace. Nejprve je ale vhodné stručně popsat jazyk UML.

UML

UML (informace čerpány z [17]), Unified Modeling Language, je v softwarovém inženýrství grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů. UML nabízí standardní způsob zápisu jak návrhů systému včetně konceptuálních prvků jako jsou business procesy a systémové funkce, tak konkrétních prvků jako jsou příkazy programovacího jazyka, databázová schémata a znovupoužitelné programové komponenty.

UML podporuje objektově orientovaný přístup k analýze, návrhu a popisu programových systémů. UML neobsahuje způsob, jak se má používat, ani neobsahuje metodiku/y, jak analyzovat, specifikovat či navrhovat programové systémy. Standard UML definuje standardizační skupina Object Management Group (OMG).

Vývoj UML začal v roce 1994, kdy Grady Booch a Jim Rumbaugh začali ve firmě Rational Software (nyní součást firmy IBM) spojovat své metodiky – Booch a OMT (Object Modeling Technique). Na konci roku 1995 do firmy Rational Software vstoupil Ivar Jacobson se svojí metodologii OOSE (Object-Oriented Software Engineering). Výsledkem jejich práce byl návrh UML (verze 0.9) a metodika RUP (Rational Unified Process). Standardizační organizace OMG v roce 1997 přijala jako standard UML verze 1.1, ve které byly začleněny prvky z dalších metodik (označení UML 1.0 se používá pro návrh, který poslala firma Rational Rose standardizační komisi). Postupně se upřesňovala specifikace a vznikaly další verze 1.2 (1998), 1.3 (1999), 1.4 (2001) a 1.5 (2002). Větší změny byly začleněny do verze 1.3. Od roku 2001 OMG připravovala verzi 2.0, která přináší podstatná rozšíření. Verze 2.0 byla uvolněna v červenci r. 2007. Nynější nejnovější verze je 2.2 z února roku 2009.

Způsobů použití UML je mnoho. Tento jazyk lze použít jako podpůrný nástroj pro komunikaci mezi vývojáři pro zaznamenání myšlenek a návrhů, nebo pro zaznamenání kompletního návrhu či realizace. UML lze rovněž použít k vygenerování přímo spustitelného kódu, což ovšem vyžaduje specializované nástroje a přesné vyjadřování v UML diagramech.

Popis UML uzavírá přehled diagramů UML 2.0 včetně jejich rozčlenění do skupin:

- diagramy struktury
    - diagram tříd, diagram komponent, diagram vnitřní struktury, digram nasazení, diagram balíčků, diagram objektů (též instancí)
- diagramy chování
    - diagram aktivit, diagram případů použití, stavový diagram
    - diagramy interakce
        - sekvenční diagram, diagram komunikace, diagram přehledu interakcí, diagram časování

5.3.1 Diagram tříd

Diagram tříd představuje zobrazení statické struktury systému prostřednictvím tříd a vztahů mezi nimi. Kromě tříd může obsahovat také různá rozhraní, balíčky nebo seskupení. Základními prvky diagramu tříd jsou:

Třída je kategorie nebo skupina objektů, které mají podobné atributy a obecné chování. Třída je reprezentována pojmenovaným obdélníkem s oddělenými prostory pro atributy a operace. Objekt je potom instancí třídy – specifická věc se specifickými hodnotami atributů a chováním.

Vztahem se nazývá spojení mezi dvěma třídami. Znázorňuje se jako čára spojující dvě třídy, nad ní se pak nachází název asociace. Pokud má třída vztah k další třídě, obvykle hraje ve vztahu jistou roli. Vztah může být také více komplexní a k jedné třídě se může pojit více tříd.

Vícenásobný vztah je zvláštním druhem asociace, která uvádí počty objektů tříd, která jsou ve vzájemném vztahu.

Dědičnost – Třída může dědit ze své mateřské třídy atributy a operace. Mateřská třída je definována obecněji než třída, která od ní dědí. Vztah dědičnosti se v UML zapíše čárou, která spojuje třídu rodiče a potomka. Na straně rodičovské třídy je ukončena prázdným trojúhelníkem.

Abstraktními třídami se nazývají třídy bez vlastních objektů. Na tuto vlastnost lze upozornit zápisem názvu třídy kurzívou.

Agregace je zvláštním druhem vztahu, kdy se vlastní třída skládá za skupiny komponentních tříd. Agregace je reprezentovaná jako hierarchie, kde na vrcholu se nachází třída, která představuje celek, a pod ní komponenty této třídy. Na straně třídy celku je vztah ukončen kosočtvercem, od kterého vedou spojovací čáry ke komponentním třídám.

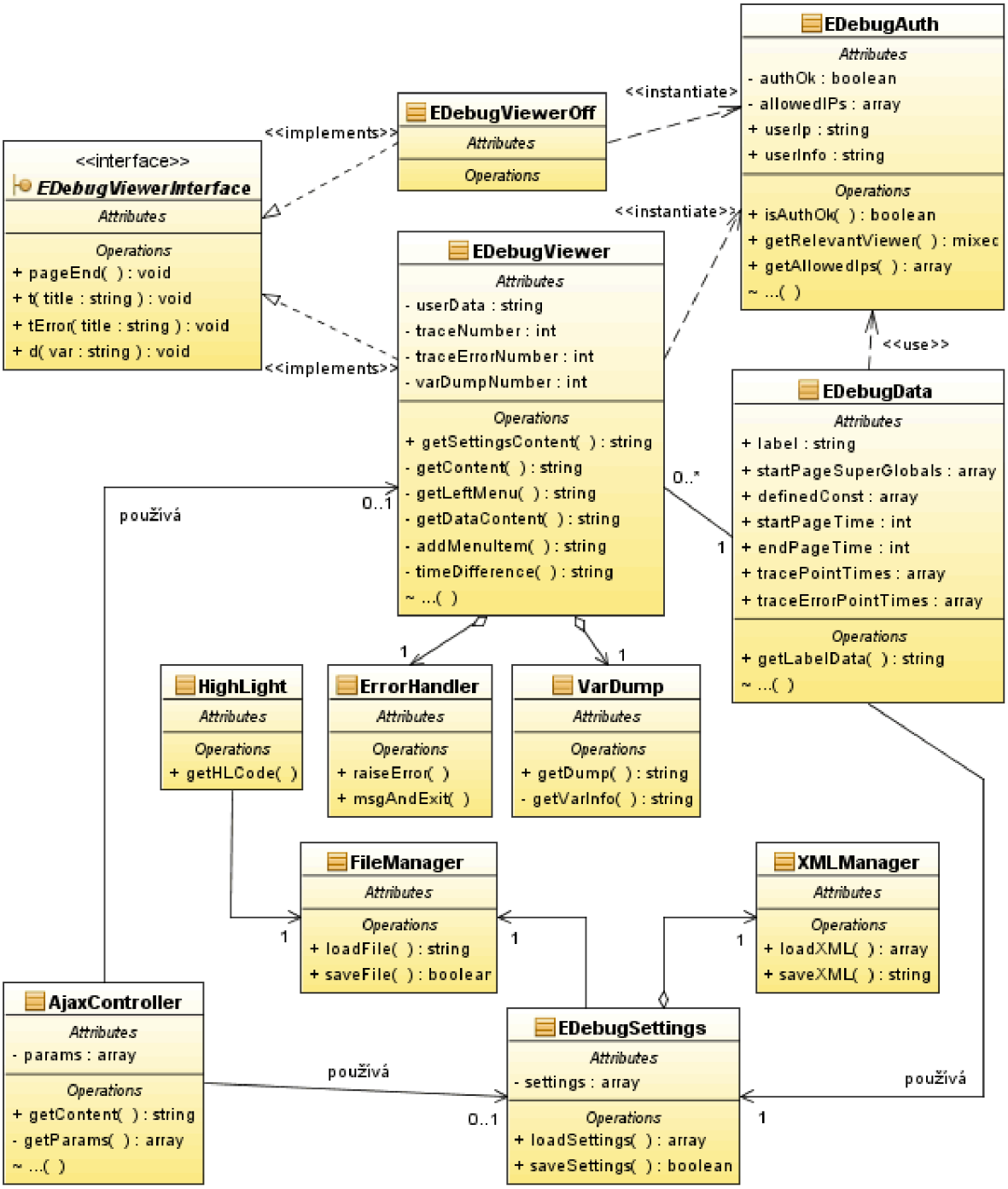
Kompozice je typem silné agregace. Každá komponenta v kompozitu může náležet pouze jednomu celku. Značení je stejné jako u agregace, pouze kosočtverec je vyplněný.

Rozhraní je množina operací, které specifikují určité hledisko chování třídy, a svou množinu operací třída prezentuje ostatním třídám (vyváží). Rozhraní se modeluje podobně jako třída: má obdélníkovou ikonu, seznam operací, ale postrádá vlastní atributy. Pro rozlišení rozhraní od tříd, se používá klíčové slovo `<<interface>>` umístěné nad názvem rozhraní.

Viditelnost je aplikována na atributy a operace pro určení, v jakém rozsahu ostatní třídy mohou používat atributy nebo operace dané třídy. Viditelnost se uvádí před názvy atributů a operací a značí se: „+“ – veřejná, „-“ – soukromá, „#“ – chráněná.

Diagram tříd nástroje této práce je zobrazen na obrázku Obrázek 4-3. Popis jednotlivých entit diagramu na obrázku je následující:

EDebugViewerInterface – rozhraní, které deklaruje uživatelské metody k použití ve zdrojovém kódu laděné aplikace. Pomocí těchto metod je prováděna interakce zdrojovým kódem popsaná v podkapitole 5.1.1.



Obrázek 4-3: Diagram tříd

EDebugViewerOff – Třída implementující rozhraní EDebugViewerInterface. Instance této třídy je vytvářena objektem EDebugAuth, pokud aplikaci používá uživatel, který nemá oprávnění používat ladící nástroj. Obsahuje pouze prázdné metody deklarované v EDebugViewerInterface.

EDebugViewer – Stěžejní třída celého nástroje. Zajišťuje zobrazení obsahu nástroje a používá k tomu ostatních tříd. Je to vstupní bod uživatelské interakce s nástrojem.

EDebugAuth – Třída zajišťující autentizaci a autorizaci uživatele. Objekt této třídy vytváří instance tříd EDebugViewer a EDebugViewerOff podle úspěšné či neúspěšné autentizace.

EDebugData – Třída uchovávající různá data nejen o aplikaci, které nástroj používá. S objektem této třídy pracuje objekt EDebugViewer.

HighLight – Zajišťuje obarvení zdrojového kódu, používá k tomu objekt třídy FileManager pro načtení obsahu souboru.

ErrorHandler – Tato třída je část třídy EDebugViewer a má na starosti zachycení a zpracování výstupu chyb.

VarDump – Rovněž část třídy EDebugViewer, která zpracovává výpis obsahu proměnné.

FileManager – Třída sloužící k načítání a ukládání souborů.

EDebugSettings – Tato třída obstarává ukládání a načítání konfigurací nástroje. Používá k tomu tříd XMLManager a FileManager.

XMLManager – Slouží ke generování XML kódu a získávání hodnot z XML kódu. Je částí třídy EDebugSettings.

AjaxController – Tato třída zpracovává AJAX požadavky. K ukládání nastavení používá třídu EDebugSettings a k získání výstupu používá třídu EDebugViewer.

5.3.2 Vzor softwarové architektury, třídy z pohledu vrstev

Koncepce nástroje využívá *vzor softwarové architektury* s označením *MVC (Model-View-Controller)*. Pro další pokračování návrhu je vhodné si tyto pojmy krátce objasnit (čerpáno z [18]).

Vzor softwarové architektury

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

Hlavním aspektem vzorů softwarové architektury je ztělesnění atributu kvality řešení. Je důležité ve fázi návrhu správně zvolit tento vzor na základě požadovaných vlastností.

Z nejznámějších takových vzorů jmenujme např. Model-View-Controller, Three-tier, Presentation-abstraction-control, Pipeline nebo Peer-to-Peer.

MVC

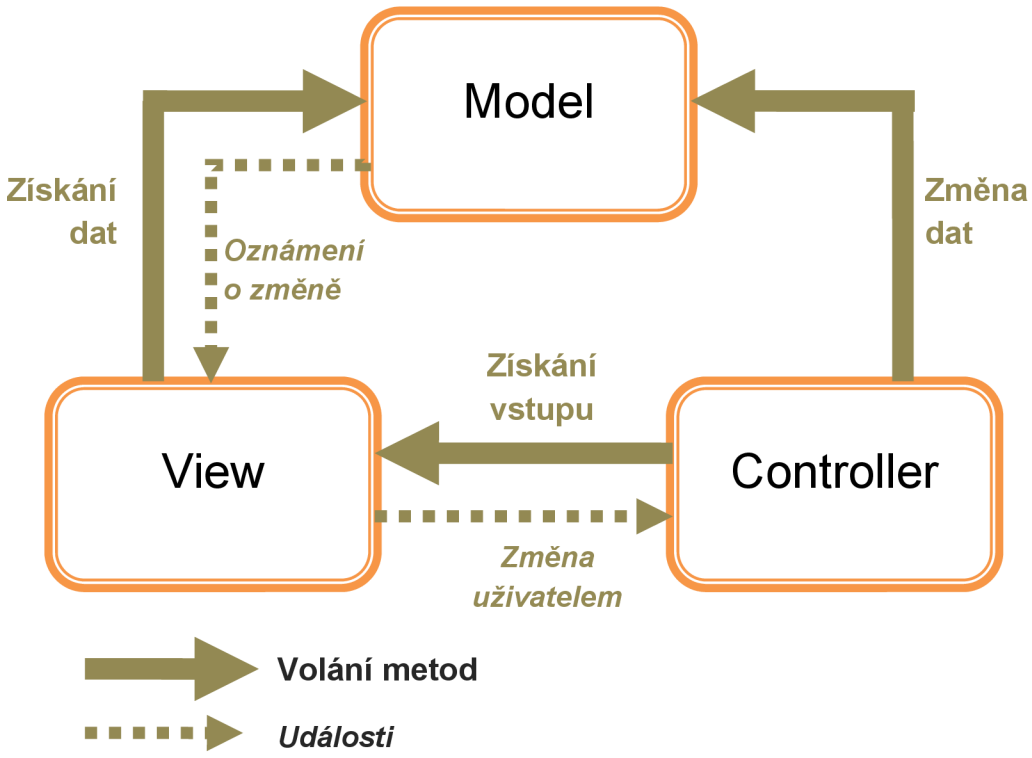
MVC – Model-View-Controller je jeden z nejpoužívanějších vzorů pro webové aplikace. Je označován také jako *Model-2*. Odděluje datový model aplikace, uživatelské rozhraní a řídící logiku tak, aby modifikace jedné složky měla minimální řídící vliv na složku druhou. Poprvé byl popsán roku 1979 Trygve Reenskaugem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

Jak název napovídá, vzor MVC se skládá ze tří vrstev:

- **Model** představuje doménově specifickou reprezentaci informací, což u webových aplikací znamená především získání požadovaných dat z databáze (což se ovšem netýká nástroje této práce, ten získává různá data ze serveru, aplikace a souborů) 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.

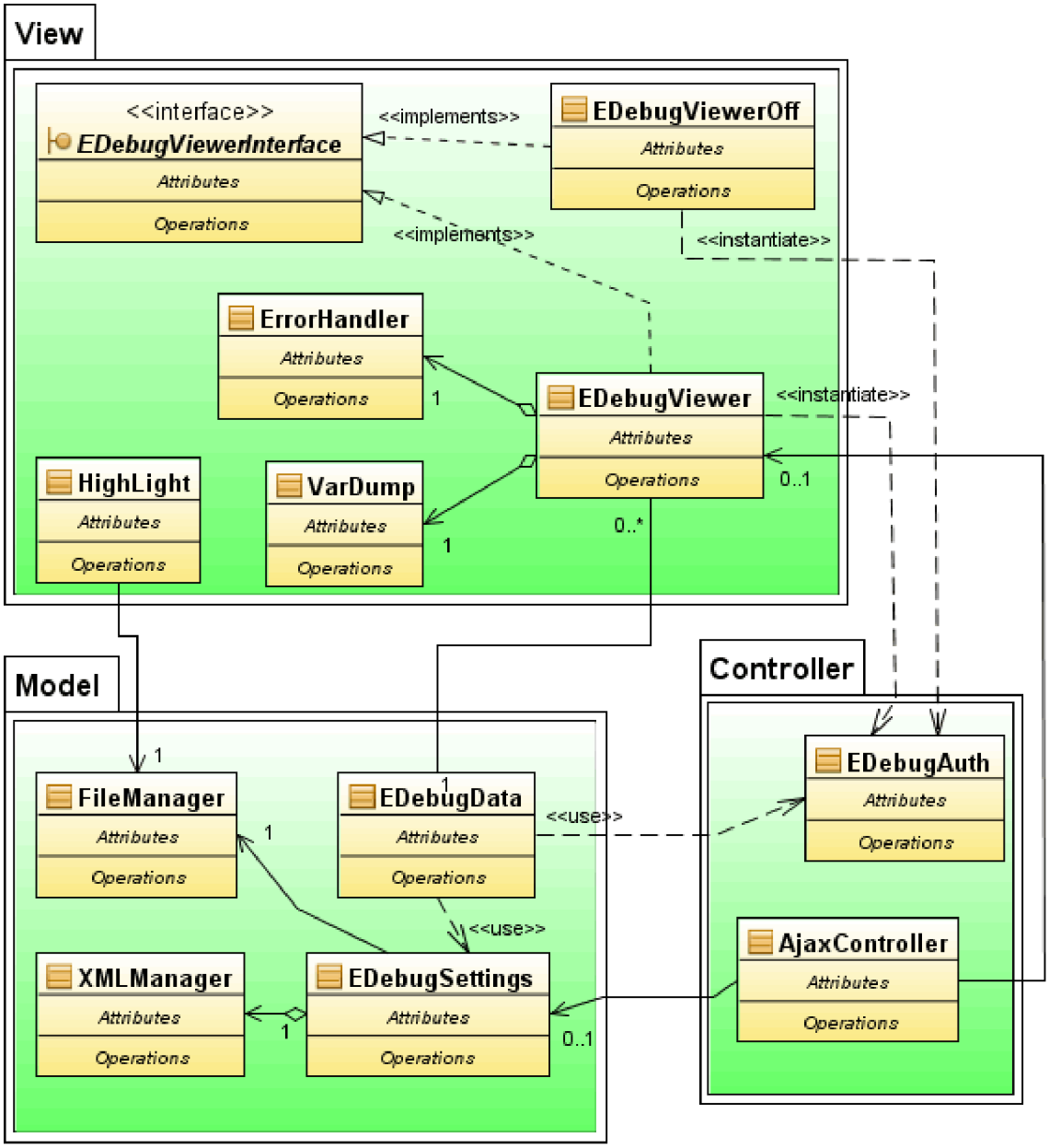
Obecně je model nezávislý na pohledu, tedy pohled posílá požadavky modelu, který na ně reaguje. Může existovat i zpětná komunikace, kdy model informuje pohled o případných změnách, které nastaly. Řadič přijímá vnější požadavek a rozhoduje o akci, která se má provést, načež pošle zprávu modelu. Ten na základě této zprávy připraví data, která jsou požadována pro daný typ zprávy. Řadič zasílá zprávu zároveň i pohledu, který sestaví šablonu pro zobrazení na výstupu a jakmile je šablona připravena, posílá pohled zprávu modelu. Na základě této zprávy jsou získána data, která doplňují šablonu. Výsledek je zobrazen na výstup.

Vzor MVC znázorňuje obrázek Obrázek 4-4.



Obrázek 4-4: Vzor Model-View-Controller

Následující obrázek Obrázek 4-5 znázorňuje umístění tříd nástroje z pohledu jednotlivých vrstev vzoru MVC.



Obrázek 4-5: Umístění tříd nástroje z pohledu jednotlivých vrstev vzoru MVC

5.4 Chování

Tato podkapitola se věnuje modelování dynamické struktury aplikace nástroje. Obsahuje diagramy chování a interakce pouze vybraných případů použití netriviálního charakteru. Triviálnější případy použití jsou obdobou či podmnožinou těch složitějších, které jsou diagramy zobrazeny.

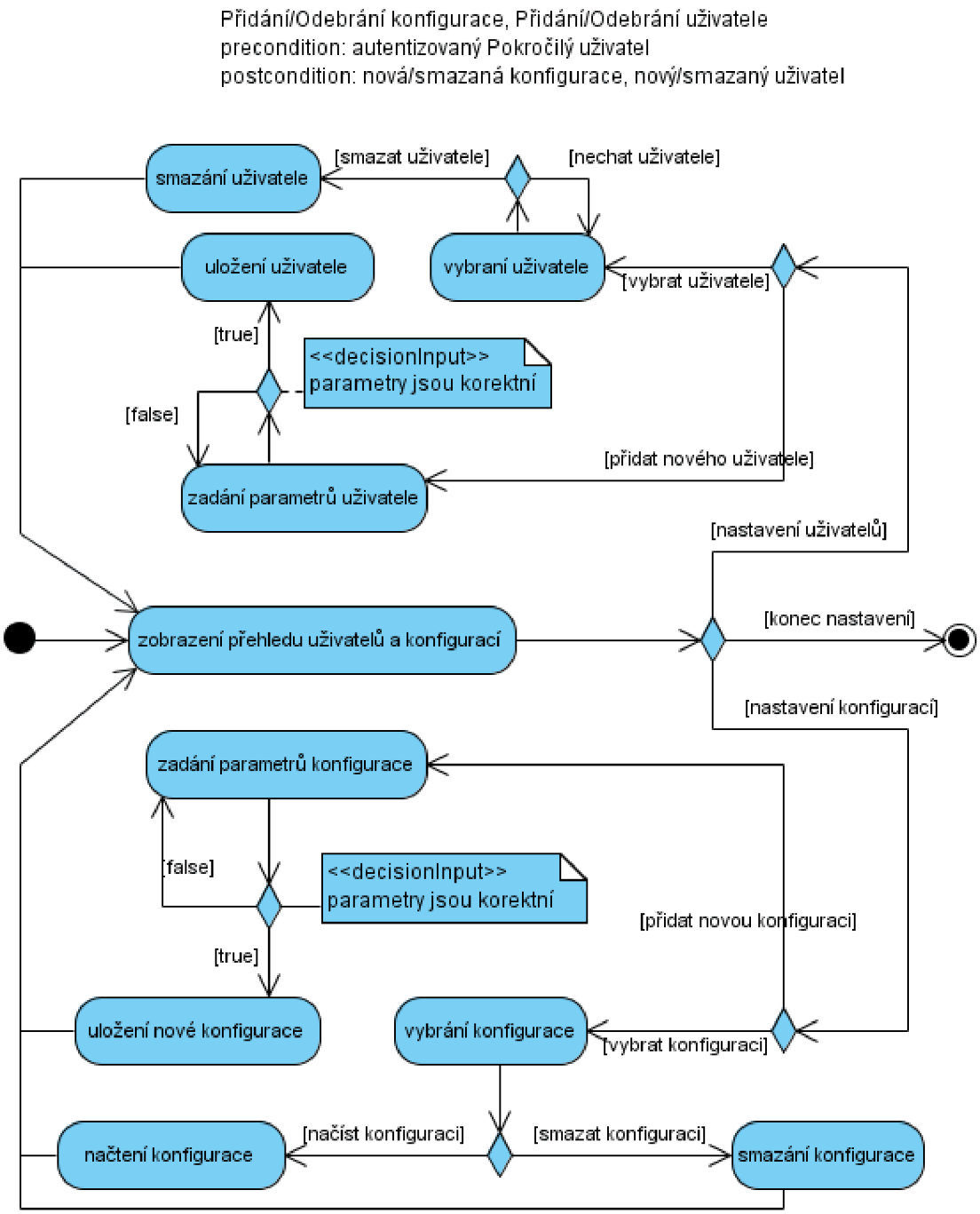
5.4.1 Diagram aktivit

Jedná se o zvláštní případ stavového diagramu, kdy jednotlivé stavy představují aktivity a dokončením těchto aktivit jsou volány přechody. V tomto diagramu jsou zdůrazněny samotné

aktivity. Diagram aktivit obsahuje především aktivity, rozhodovací uzly a větve. Popisuje řazení aktivit s podporou sekvenčního a paralelního řazení. Diagram je vhodný pro pochopení toku činností a odvození jeho řešení. Je užitečný při analýze a implementaci obchodních procesů, operací a případů použití, protože je přímo navržen jako zjednodušený pohled na dění v průběhu procesů a operací. Lze jej také použít při řešení vícevláknových aplikací.

Podpora paralelního chování umožňuje modelování toku činností a vícevláknového programování. Nevýhodou je nejasná vazba mezi objekty a aktivitami. Diagram není vhodný pro reprezentaci složitého větvení, popis spolupráce objektů a chování objektu v průběhu jeho života.

Diagram aktivit zahrnující případy použití „Přidání/Odebrání uživatele“ a „Přidání/Odebrání konfigurace“ je zobrazen na obrázku Obrázek 4-6.



Obrázek 4-6: Digram aktivit zahrnující příp. použití Přidání/Odebrání konfigurace/uživatele

5.4.2 Diagramy interakce

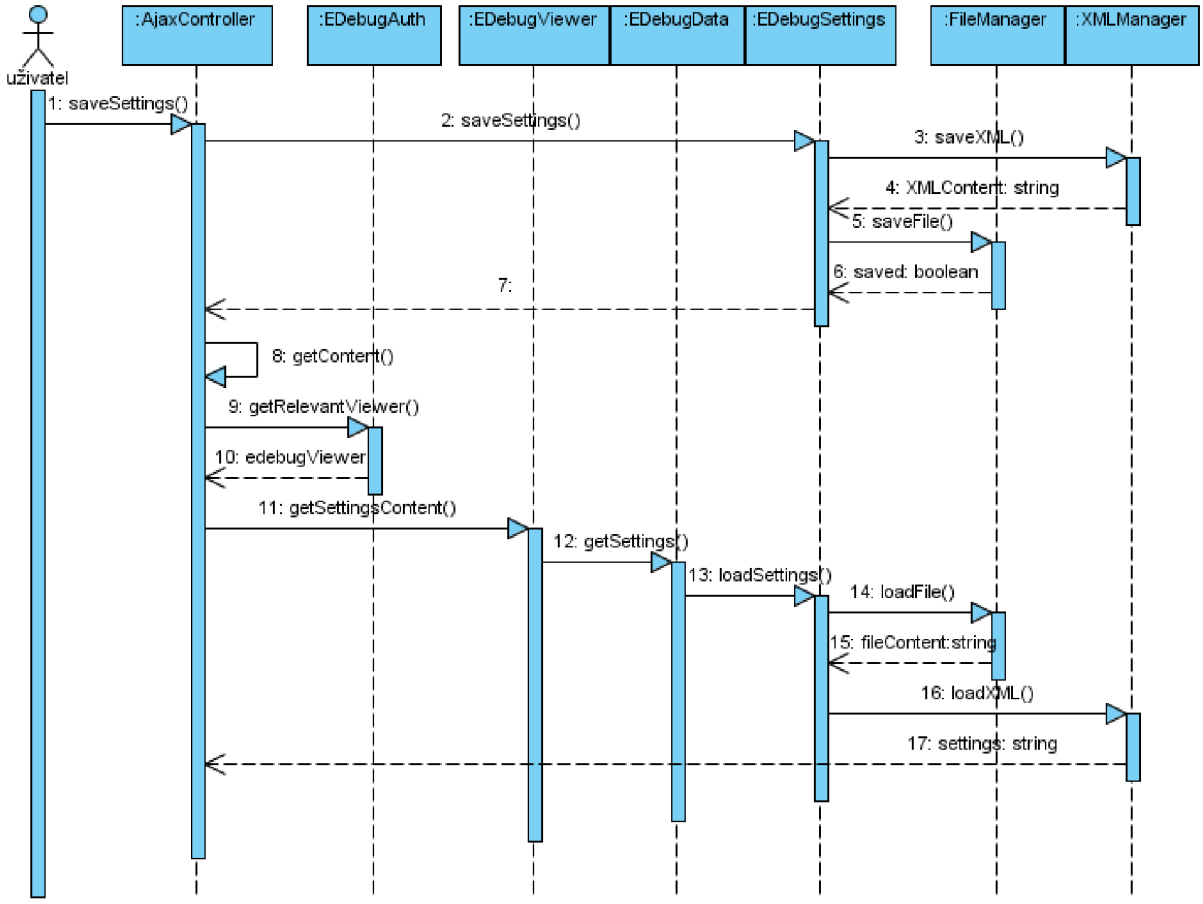
Diagramy interakce popisují spolupráci skupin objektů pro dosažení určitého chování – typicky zachycují chování jednoho případu použití.

Diagram sekvence

Sekvenční diagram identifikuje základní vnitřní dynamiku aplikace a potřebné metody jednotlivých tříd. Diagramy interakce popisují spolupráci skupiny objektů za účelem specifického chování. Sekvenční diagramy patří mezi nejčastěji používané diagramy interakcí. Tento typ diagramu popisuje chování objektů v rámci jednoho scénáře.

Sekvenční diagramy ukazují, jak objekty komunikují navzájem mezi sebou v časové rovině. To zahrnuje rozšíření pohledu na systém o další důležitou proměnnou, kterou je čas. Interakce mezi objekty probíhá v určité posloupnosti, a tato posloupnost proběhne od začátku do konce za určitý časový interval. Sekvenční diagramy nejsou vhodné pro zobrazení detailů algoritmu a precizní definici chování, ale podávají jasný obraz o činnosti několika účastníků během modelované interakce daného případu použití. Samotný sekvenční diagram se skládá především z objektů a zpráv.

Digram sekvence zobrazující případ použití „Uložení nastavení“ je zobrazen na obrázku Obrázek 4-7.



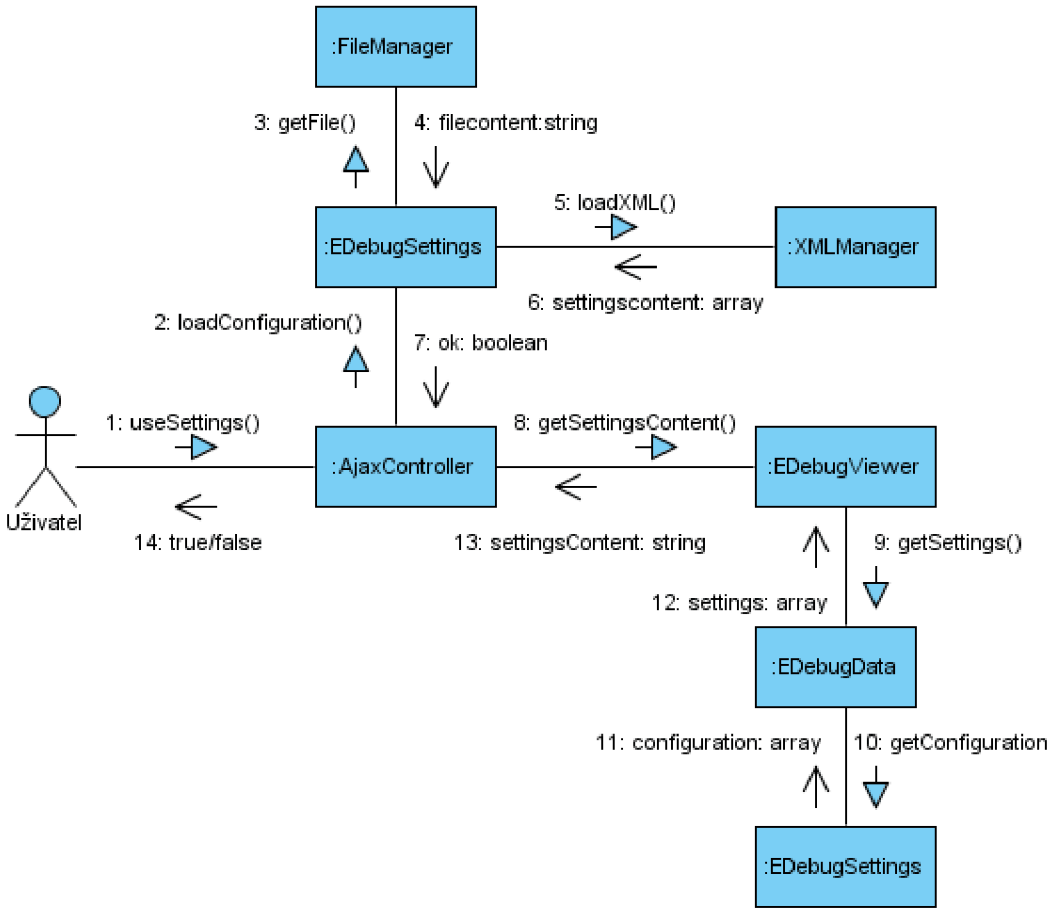
Obrázek 4-7: Diagram sekvence znázorňující případ použití „Uložení nastavení“

Diagram komunikace

Posledním návrhovým diagramem této práce je Diagram komunikace je další z diagramů interakce, který ukazuje účastníky interakce a datová spojení mezi nimi. Ukazuje jiným způsobem podobnou informaci jako sekvenční diagram. Diagram komunikace dovoluje volné umísťování účastníků, jejich propojování čarami spojení a pomocí číslování zobrazovat sekvence zpráv. Platí zde pravidlo,

že diagram komunikace je vhodnější při zobrazení spojení a na druhou stranu sekvenční diagram je vhodnější pro zobrazení sekvencí volání.

Následný diagram komunikace na obrázku Obrázek 4-8 znázorňuje případ použití „Použij konfiguraci“.



Obrázek 4-8: Diagram komunikace znázorňující případ použití „Požij konfiguraci“

Kapitola 6

Implementace

Tato kapitola se již věnuje samotné implementaci nástroje. Cílem ovšem není popsat veškeré funkce a části nástroje, ale vyzdvihnout ty, které jsou důležité nebo zajímavé. V případě potřeby detailnějších informací je možné nahlédnout do zdrojových kódů.

6.1 Adresářová struktura

Adresářová struktura výsledné aplikace nástroje je následující:

```
..
  |-class
  |-css
  |-include
  |-interface
  |-js
  |-settings
    |-saved_edebug_configs
```

Stručný popis jednotlivých adresářů:

`..` V kořenovém adresáři se nachází pouze jeden soubor – inicializační soubor nástroje `init.php`.

`class` V tomto adresáři se nachází všechny soubory tříd, které nástroj používá. Soubory tříd mají specifickou koncovku „.class.php“.

`css` Adresář obsahující soubory kaskádových stylů.

`include` V tomto adresáři se vyskytují různé pomocné soubory, které jsou do kódu vkládány PHP funkcí `include`.

`interface` Interface je adresář obsahující soubory s rozhraními, které nástroj používá. Soubory rozhraní mají specifickou koncovku „.interf.php“.

`js` Obsahuje skripty v jazyce javascript.

`settings` Adresář, který obsahuje soubory s nastavením nástroje. Jednak obsahuje soubor ukládající informace o uživatelích a jejich přístupových právech, a jednak obsahuje soubory s uloženými konfiguracemi nástroje.

`|-saved_edebug_configs` Tento adresář je umístěn v adresáři `settings` a obsahuje jednotlivé soubory s uloženými konfiguracemi nástroje.

V nadřazeném adresáři se nachází kód laděné aplikace. Fyzický obsah nástroje této práce se potom do aplikace vkládá nahráním výše uvedené adresářové struktury umístěné do jednoho adresáře do adresářové struktury aplikace.

6.2 Popis některých důležitých skriptů a souborů

init.php

`Init.php` je vstupním bodem k vložení aplikace nástroje do aplikace, kterou chce uživatel ladit. Jedná se o inicializační soubor, který vytváří hlavní objekty nástroje a definuje potřebné konstanty.

Tento skript začíná definováním konstant, které bude nástroj využívat. Nastavuje konstanty jako relativní a absolutní cestu od kořene serveru nebo verzi nástroje.

Dále skript obsahuje speciální funkci jazyka PHP pro načtení souborů tříd, pokud je třída volána. Tato funkce je popsána dále v podkapitole 6.3.1.

Na závěr následuje vykonání autentizace a autorizace vytvořením objektu třídy `EDebugAuth` a podle její úspěšnosti či neúspěšnosti je vytvořena instance třídy zapnutého nástroje (`EDebugViewer`) nebo vypnutého nástroje (`EDebugViewerOff`).

EDebugViewer.class.php

Tento soubor obsahuje hlavní a nejobsáhlejší třídu nástroje `EDebugViewer`, která implementuje zobrazení dat nástroje a rovněž uživatelské metody.

Jelikož třída `EDebugViewer` implementuje rozhraní `EDebugViewerInterafce`, je toto vkládáno (`funkcí require_once()`) na začátku souboru.

Kód třídy po deklaraci atributů začíná metodou, která představuje konstruktor, ve které jsou vytvářeny potřebné objekty a nastaven začátek odpočítávání doby vykonávání skriptu.

Následuje definování uživatelských metod, které může uživatel nástroje volat ze svého skriptu své vyvíjené aplikace. Mimo tyto metody je definována metoda, která je volána na konci každého skriptu, který je určen pro ladění, jenž se stará vytvoření HTML kódu nástroje a jeho zobrazení na výstupu.

V další části kódu jsou uvedeny veřejné metody, které používají jiné objekty tříd.

Zbylá část kódu třídy obsahuje privátní metody využívané ke generování kódu a upravování a zpracování výstupu.

EDebugData.class.php

Jak název napovídá, `EDebugData.class.php` je soubor obsahující implementaci třídy `EDebugData`, která slouží k získávání a uchování dat využívaných nástrojem.

Po deklaraci atributů třída obsahuje konstruktor, který nastavuje některé hodnoty proměnných, jako např. obsah všech superglobálních proměnných.

Následují převážně funkce pro nastavení a získání hodnot atributů, hovorově zvané „getry“ a „setry“ (čteno česky).

Dále tato třída obsahuje funkce pro měření a zpracování času, které jsou používány k uchovávání časových hodnot začátku a konce provádění skriptu nebo vzniku trasovacího bodu.

Konečná část kódu třídy obsahuje privátní pomocné metody. Jedna z metod je i metoda pro získání hodnot superglobální proměnné, která je popsána dále v podkapitole 6.3.3.

Easydebug.js

Tento soubor obsahuje kód jazyka Javascript. Tvoří důležitou část nástroje a některé jeho prvky plní funkci řadiče popsaného v podkapitole 5.3.

Soubor provádí dynamické vložení kaskádových stylů do stránky, a dynamické umístění obsahu nástroje na začátek stránky. Rovněž obsahuje implementaci technologie AJAX.

Obsah souboru začíná vložením kaskádových stylů do stránky a přidáním události ke správci událostí spouštěných po načtení, která je volána po načtení stránky.

Dále obsahuje funkce pro dynamické umístění obsahu nástroje na začátek stránky, schování či zobrazení HTML elementu nebo zvýraznění či „odzvýraznění“ HTML elementu.

Poslední část souboru obsahuje kód, který implementuje technologii AJAX.

permission.php

Tento soubor uchovává informace o uživatelích nástroje a je načítán při každém požadavku na zobrazení nástroje.

Je složen ze dvou hlavních částí. První část obsahuje definici pole, které uchovává informace o povolených IP adresách a jejich popisu.

Druhá část obsahuje definici pole, které uchovává informace o IP adresách, které mají vyšší práva, tj. práva Pokročilého uživatele (viz požadavky na nástroj).

Kód obsahuje speciální komentáře, které slouží jako startovací a koncové značky při zpracování souboru při přidávání/odebírání uživatelů. Tyto značky jsou tvaru „`// START`“ a „`// END`“.

6.3 Zajímavé úseky kódu

6.3.1 Vkládání souborů s třídami

Ke vkládání souborů, které obsahují potřebné třídy, které jsou volány, slouží speciální funkce jazyka PHP `__autoload()`:

```
function __autoload($classname)
{
      require_once EDEBUG_DIR_SERVER.'/class/'.$classname.'.class.php';
}
```

Tato funkce funguje následovně: Pokud je ve skriptu volána jakákoliv třída, která není ve skriptu definována, PHP zjistí existenci funkce `__autoload()` a pokud existuje, provede její obsah, přičemž jako parametr funkce je název volané třídy. Takto je zajištěno vkládání potřebných souborů tříd bez nutnosti toto explicitně obstarávat.

6.3.2 Získání hodnot polí a objektů

Vytváření výstupních informací v případě proměnné typu objekt:

```
if (is_object ($var)) {
      $class = get_class ($var);
      if (substr_count($class, 'EDebug') > 0) {
          $out .= "<span style='color:#888a85;'><b>Object</b>(<i>$class</i>) object of Eas…
      }else{
          $out .= "<span style='color:#AC0042;'><b>Object</b></span>(<i>$class</i>)";
          $parent = get_parent_class ($var);
          $out .= $parent != '' ? " <span style='color:purple;'>extends</span> <i>$parent<…
          $out .= " (\n";
          $arr = get_object_vars ($var);
          while (list($prop, $val) = each($arr)) {
              $out .= "$spc  " . "'$prop' <span style='color:#888a85;'>=></span> ";
              if ($indent > 15 ) {
                  $out .= "max indent reached\n";
              }else{
                  $out .= $this->getVarInfo($val, $name != '' ? $prop : '', $indent + 1);
              }
          }
          // show methods
          $arr = get_class_methods ($var);
          $out .= "$spc  " . "$class methods: " . count ($arr) . " ";
          if (in_array ($class, $methods)) {
              $out .= "[already listed]\n";
          } else {
              $out .= "(\n";
              $methods[] = $class;
              while (list($prop, $val) = each($arr)) {
                  if ($val != $class) {
                      $out .= $indent_chars . "$spc  " . "->$val();\n";
                  } else {
                      $out .= $indent_chars . "$spc  " . "->$val(); [<b>constructor</b>]\n…
                  }
              }
              $out .= "$spc  " . ")\n";
          }
          $out .= "$spc)";
      }
}
```

Při vytváření výstupních informací dochází k rekurzivnímu volaní metody `getVarInfo()`, která vytváří výstupní informace k získání hodnot polí a objektů.

6.3.3 Získání hodnoty superglobální proměnné

Funkce pro získání hodnot superglobální proměnné, jejíž název je uveden v parametru funkce:

```
public function getEndSG($type_of_SG) {
```

```
        if(!in_array($type_of_SG, array_keys($this->SuperglobalsTypes))){
            return null;
        }
        if($type_of_SG == 'globals'){
            return array_diff_key($GLOBALS,array_flip($this->SuperglobalsTypes));
        }else{
            $SGname = $this->SuperglobalsTypes[$type_of_SG];
            global ${$SGname};
            if(isset(${$SGname})) {
                return ${$SGname};
            }else{
                return null;
            }
        }
}
```

Proměnná `$this->SuperglobalsTypes` obsahuje výčet názvů superglobálních proměnných. Zajímavá část v tomto kódu je získání hodnoty superglobální proměnné, kdy je použita tzv. „double-dollar“ notace pro získání hodnoty proměnné s názvem hodnoty proměnné viz

```
global ${$SGname};
```

6.3.4 Vyznačení řádku ve zdrojovém kódu

Způsob načtení souboru a obarvení syntaxe:

```
ob_start();
highlight_file($file);
$data = ob_get_contents();
ob_end_clean();
```

Obarvení probíhá pomocí PHP funkce `highlight_file()`, která značné usnadňuje práci.

6.3.5 Zpracování chyb

Způsob zpracování fatálních chyb byl ve starších verzích PHP poněkud značný problém. Od verze PHP 5.2 byla do PHP přidána funkce `error_get_last()`, které zpracování fatálních chyb výrazně ulehčuje. Funkce nástroje pro zpracování všech chyb včetně fatálních je následující:

```
function raiseError($errno, $errstr, $errfile, $errline) {
      if ( !defined('E_STRICT') ) define('E_STRICT', 2048);
      $errortype = array (
          E_ERROR          => "Error",
          E_WARNING        => "Warning",
          E_PARSE          => "Parsing Error",
          E_NOTICE          => "Notice",
          E_CORE_ERROR      => "Core Error",
          E_CORE_WARNING    => "Core Warning",
          E_COMPILE_ERROR  => "Compile Error",
          E_COMPILE_WARNING => "Compile Warning",
```

```
            E_USER_ERROR      => "User Error",
            E_USER_WARNING    => "User Warning",
            E_USER_NOTICE     => "User Notice",
            E_STRICT          => "Runtime Notice"
        );
        if ( !in_array($errno,array_keys($errortype)) ) {
            $this->edebugViewer->tError( "index '$errno' neni v seznamu \$errortype…
        }
        $html_str = "<pre>$errstr</pre>\n<br />{$errortype[$errno]} ({$errno}): in …
        $errstr = preg_replace('/(\r\n|\n)+$/', '', $errstr);
        $simple_msg_to_trace = "<span style=\"color: red;\"><b>{$errortype[$errno]}…
        $simple_msg  =  "<span  style=\"color:  red;\"><b>{$errortype[$errno]}</b></span><b>:
$errstr</b> in line <b>{$errline}<…
        switch ($errno) {
            case E_ERROR:
            case E_PARSE:
            case E_CORE_ERROR:
            case E_COMPILE_ERROR:
            case E_USER_ERROR :
                $this->msgAndExit( $html_msg );
                break;
            case E_NOTICE:
                $this->edebugViewer->tError( $simple_msg_to_trace);
                echo $simple_msg;
                break;
            case E_STRICT:
                $this->edebugViewer->tError( $simple_msg_to_trace );
                echo $simple_msg;
                break;
            default:
                $this->edebugViewer->tError( $simple_msg_to_trace );
                echo $simple_msg;
                break;
        }
}
```

Uvedená funkce je registrována jako funkce pro zpracování chyb, díky voláním:

```
set_error_handler(array(&$this->errorHandler, 'raiseError'));
register_shutdown_function(array(&$this->errorHandler, 'raiseError'));
```

6.3.6 Dynamické umístění oblasti nástroje

Následují kód jazyka Javascript slouží k dynamickému umístění oblasti nástroje na začátek stránky:

```
function createEDebugDiv() {
      var bodyNode = document.getElementsByTagName("body")[0];
      //main div
      var MainDivNode = document.getElementById('edebug_maindiv');
      bodyNode.insertBefore(MainDivNode,bodyNode.firstChild);
      MainDivNode.style.visibility = 'visible';
}
```

Hlavní oblast nástroje má identifikátor hodnoty „edebug_maindiv“, která je pomocí této hodnoty nalezena a přemístěna hned za počáteční HTML značku stánky `<body>`.

Kapitola 7

Testování a nasazení do provozu

Testování probíhalo převážně v průběhu vytváření nástroje a obnášelo ověřování funkčnosti každé nově přidané funkcionality. Během tohoto procesu jsem proto testoval správné chování nových prvků ve všech stavech, do kterých se mohou dostat. Největší zaměření bylo kladeno na extrémní (nejkritičtější) stavy. Nově přidané funkce nástroje mohly ovlivnit chování těch, které byly již dříve v této fázi testovány. Z tohoto důvodu jsem s každou nově přidanou funkcionalitou, která aspoň z části ovlivňovala již ověřené funkce, tyto znovu testoval.

K testování bylo vytvořeno několik testovacích skriptů se zaměřením na prověření vždy určité funkcionality nástroje. Tyto testovací skripty jsou součástí ukázkové aplikace příkladů použití. Testovací skripty ověřují funkcionality nástroje, jako jsou:

- Zobrazení hodnot superglobálních proměnných a konstant.
- Doba vygenerování skriptu.
- Trasování a čas mezi dvěma trasováními.
- Zachycení a zpracování chyb, vytváření uživatelských chyb.
- Výpis hodnot a struktury proměnných různého typu.
- aj.

Program nástroje byl rovněž zprovozněn na serveru IS VUT a nyní (květen 2009) zde běží zkušební verze, která je postupně testována. Vzhledem k rozsáhlosti a složitosti IS VUT vyžaduje však testování a úplná a finální integrace do systému dlouhodobější proces.

Nástroj práce bude rovněž po dobíhajícím testování uvolněn jako open source projekt pod názvem *EasyDebug*.

Kapitola 8

Závěr

Výsledkem této práce je teoretická analýza hlavních pojmů řešené problematiky, analýza a specifikace obecných požadavků, konkrétních požadavků na výsledný nástroj, seznámení se a popis některých nástrojů pro usnadnění vývoje a testování webových aplikací a návrh finální podoby nástroje pro usnadnění vývoje a testování PHP aplikací.

V oblasti implementační je potom výsledkem práce webový softwarový produkt, který používá a je implementován za pomocí technologií PHP, Javascript, CSS, HTML a dalších. Tento produkt slouží výhradně vývojářům webových aplikací k usnadnění vývoje a testování těchto aplikací.

K nynější době (květen 2009) se mi nepodařilo nalézt jiný obdobný nástroj, který by samotný obsahoval takové množství funkcionalit stejného zaměření a byl dostupný jako volně šiřitelný open source projekt. Proto si dovolím říci, že co se týče oblasti působnosti a s ohledem na možnosti nástroje, je tento nástroj jeden z mála svého druhu.

Tento diplomový projekt splňuje do jisté míry všechny stanovené požadavky a body zadání. Dle požadavků byla k nástroji vytvořena ukázková aplikace, byla vytvořena dokumentace a byly implementovány i některé nepovinné požadavky, jako je zobrazení seznamu všech vložených souborů do aplikace. Na druhou stranu jsou mírně potlačeny požadavky na integraci nástroje do IS VUT, i když již na serveru IS VUT běží zkušební provoz nástroje, vzhledem k rozsáhlosti a složitosti IS VUT je vyžadován dlouhodobější proces testování a finální integrace.

Jelikož bude nástroj po dokončení testovacích fází uvolněn jako volně dostupný open source projekt, je očekáváno neustálé zdokonalování a vylepšovaní jeho možností. Jako možnost vylepšení lze uvést různé specifické úpravy nástroje pro konkrétní technologie, s kterými bude používán, jako je např. zpracování dat systému pro správu verzí či modul pro sledování různých databázových spojení.

Osobním přínosem tohoto projektu pro mne bylo rozšíření si některých svých znalostí v oblasti technologií pro vývoj webových aplikací, ladících nástrojů a v dalších oblastech, které tento diplomový projekt zahrnuje. Rovněž věřím, že nástroj budu s výhodou využívat při dalším budoucím vývoji webových aplikací.

Seznam použitých zdrojů

[1] *Wikipedie: Otevřená encyklopedie: Aplikační software* [online]. c 2009. [citováno 22. 2. 2009]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Aplika%C4%8Dn%C3%AD_software>

[2] *Wikipedie: Otevřená encyklopedie: Webová aplikace* [online]. c 2009. [citováno 22. 2. 2009]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace>

[3] *PHP: Hypertext Preprocessor* [online]. c 2009. [citováno 22. 2. 2009]. Dostupný z WWW: <http://php.net/>

[4] Gutmans, A. – Bakken, S. S. – Rethans D. *Mistrovství v PHP 5*. Brno: CP Books, a.s., 2005. 656 s. ISBN 80-251-0799-X.

[5] *Linux Software: PHP debugger* [online]. c 2009. [citováno 22. 2. 2009]. Dostupný z WWW: < http://www.linuxsoft.cz/article.php?id_article=514>

[6] Pavelka, J. *Informační systém sportovního klubu*. Brno: Vysoké učení technické. Fakulta informačních technologií. Ústav informačních systémů, 2007. 50 s. Bakalářská práce. Vedoucí bakalářské práce Ing. Michael Kunc.

[7] *World Wide Web Consortium* [online]. c 2009. [citováno 22. 2. 2009]. Dostupný z WWW: <http://www.w3.org/>

[8] *Wikipedie: Otevřená encyklopedie: JavaScript* [online]. c 2009. [citováno 22. 2. 2009]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/JavaScript>

[9] Asleson, R. – Schutta, N. T. *AJAX Vytváříme vysoce interaktivní webové aplikace*. Brno: Computer Press, a.s., 2006. 272 s. ISBN 80-251-1285-3.

[10] Castagnetto, J. – Rawat, H. – Schumann, S. *PHP Programujeme profesionálně*. Brno: Computer Press, a.s., 2001. 676 s. ISBN 80-722-6310-2.

[11] Welling, L. – Thomson, L. *PHP a MySQL – rozvoj webových aplikací*. Praha: SoftPress s.r.o., 2004. 912 s. ISBN 80-86497-60-7.

[12] Schlossnagle, G. *Pokročilé programování v PHP 5*. Brno: Zoner Press, 2004. 640 s. ISBN 80-86815-14-5.

[13] Gilmore W. J. *Velká kniha PHP a MySQL 5 - kompendium znalostí pro začátečníky i profesionály*. Brno: Zoner Press, 2007. 864 s. ISBN 80-86815-53-6.

[14] Sklar, D. *PHP – moduly, rozšíření a akcelerátory*. Brno: Zoner Press, 2005. 344 s. ISBN 80-86815-19-6

[15] *PHP Classes Repository* [online]. c 2009. [citováno 21. 3. 2009]. Dostupný z WWW: <http://www.phpclasses.org/>

[16] Kosek, J. *XML pro každého – podrobný průvodce*. Praha: Grada Publishing s.r.o., 2000. 164 s. ISBN 80-7169-860-1

[17] *Wikipedie: Otevřená encyklopedie: Unified Modeling Language* [online]. c 2009. [citováno 30. 4. 2008]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/UML>

[18] Špaček, P. *Aplikační platforma v PHP*. Brno: Vysoké učení technické. Fakulta informačních technologií. Ústav informačních systémů, 2008. 51 s. Diplomová práce. Vedoucí diplomové práce Mgr. Martínek Zdeněk.

Seznam použitých zkratek a symbolů

AJAX – Asynchronous JavaScript and XML

APD – Advanced PHP Debugger

API – Application Programming Interface

ASP – Active Server Pages

CGI – Common Gateway Interface

CLI – Command Line Interface

CSS – Cascading Style Sheets

DHTML – Dynamic Hypertext Markup Language

DOM – Document Object Model

ECMA – European Computer Manufacturers Association

FI – Form Interpreter

FTP – File Transfer Protocol

GIF – Graphics Interchange Format

GUI – Grafické Uživatelské Rozhraní

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment

IMAP – Internet Message Access Protocol

IP address – Internet Protocol address

IS – Informační Systém

ISO – International Standards Organization

JPEG – Joint Photographic Experts Group

LDAP – Lightweight Directory Access Protocol

MVC – Model-View-Controller

MySQL – My Structured Query Language

ODBC - Open Database Connectivity

OMG – Object Management Group

OMT – Object Modeling Technique

OOP – Objektově Orientované Programování

PDA – Personal Digital Asistent

PEAR – PHP Extension and Application Repository

PECL – PHP Extension Community Library

PHP – PHP: Hypertext Preprocessor

PNG – Portable Network Graphics

POP3 – Post Office Protocol version 3

SMTP – Simple Mail Transfer Protocol

SNMP – Simple Network Management Protocol

SQL – Structured Query Language

UML – Unified Modeling Language

URL – Uniform Resource Locator

UTF – UCS/Unicode Transformation Format

VUT – Vysoké Učení Technické

W3C – World Wide Web Consortium

WAP – Wireless Application Protocol

WWW- World Wide Web

XHTML – Extensible Hypertext Markup Language

XML – Extensible Markup Language

Seznam příloh

Příloha A

Ostatní specifikace případů použití

Tato příloha obsahuje zbylé neuvedené specifikace případů použití z podkapitoly 3.6 z diagramu případů použití na obrázku Obrázek 3-2 na straně 30.

Případ použití: Odebrání uživatele
ID: PP8
Stručný popis: Odebrání uživatele ze systému ladícího nástroje.
Účastníci: Pokročilý uživatel
Vstupní podmínky: Autentizovaný *Pokročilý uživatel*.
Hlavní tok: 1. Systém nejprve zobrazí přehled všech dostupných uživatelů systému. 2. Případ použití se spustí, když *Pokročilý uživatel* zvolí „Odebrat uživatele“. 3. Systém odebere uživatele. 4. Systém zobrazí potvrzovací zprávu a aktualizovaný seznam uživatelů.
Následné podmínky: Žádné.
Alternativní toky: Žádné

Případ použití: Změna parametrů
ID: PP1
Stručný popis: Změna jednotlivých nastavení parametrů nástroje.
Účastníci: Uživatel
Vstupní podmínky: Autentizovaný *Uživatel*.
Hlavní tok: 1. Systém zobrazí aktuální stav parametrů. 2. Případ použití se spustí, když *Uživatel* zvolí změnu parametrů. 3. *Uživatel* nebo systém (v případě hodnoty parametru, která může nabývat jen dvou hodnot) zadá novou hodnotu parametru 4. Systém uloží novou hodnotu parametru. 5. Systém zobrazí aktualizovaný seznam parametrů.
Následné podmínky: Žádné.
Alternativní toky: Žádné

Případ použití: Zobrazení ladících hodnot
ID: PP2
Stručný popis: Zobrazení jednotlivých ladících hodnot nástroje.
Účastníci: Uživatel
Vstupní podmínky:

Autentizovaný *Uživatel*.
Hlavní tok: 1. Systém zobrazí seznam položek ke zobrazení. 2. Případ použití se spustí, když *Uživatel* vybere některou položku. 3. Systém zobrazí hodnotu vybrané položky.
Následné podmínky: Žádné.
Alternativní toky: Žádné

Případ použití: Skrytí ladících hodnot
ID: PP3
Stručný popis: Skrytí zobrazených hodnot položek nástroje.
Účastníci: Uživatel
Vstupní podmínky: Autentizovaný *Uživatel*.
Hlavní tok: 1. Systém zobrazí seznam položek ke skrytí. 2. Případ použití se spustí, když *Uživatel* vybere některou položku. 3. Systém skryje hodnotu vybrané položky.
Následné podmínky: Žádné.
Alternativní toky: Žádné

Případ použití: Vytvoření konfigurace
ID: PP4
Stručný popis: Přidání nové konfigurace do systému ladícího nástroje.
Účastníci: Uživatel
Vstupní podmínky: Autentizovaný *Uživatel*.
Hlavní tok: 1. Systém nejprve zobrazí přehled všech dostupných konfigurací. 2. Uživatel nastaví parametry konfigurace, včetně jména nové konfigurace. 3. Případ použití se spustí, když *Pokročilý uživatel* zvolí „Přidat novou konfiguraci“. 4. Pokud nejsou všechny povinné údaje vyplněny, systém vyžaduje, aby *Uživatel* zadal potřebné údaje. 5. Při správně vyplněných údajích systém uloží novou konfiguraci. 6. Systém zobrazí aktualizovaný seznam konfigurací.
Následné podmínky: Žádné.
Alternativní toky: NeplatnéÚdajeKonfigurace

Alternativní tok: Vytvoření konfigurace: NeplatnéÚdajeKonfigurace
ID: PP4.AL1
Stručný popis: Systém zruší přidávání nové konfigurace s návratem k zadávání údajů
Účastníci:

Uživatel
Vstupní podmínky: Autentizovaný *Uživatel*.
Hlavní tok: 4. Alternativní tok se spustí po kroku PP4.3 5. Systém zruší přidávání konfigurace do systému 6. Systém se vrací do kroku PP4.2
Následné podmínky: Žádné.
Alternativní toky: Žádné.

Případ použití: Načtení konfigurace
ID: PP5
Stručný popis: Načtení uložené konfigurace nástroje.
Účastníci: Uživatel
Vstupní podmínky: Autentizovaný *Uživatel*.
Hlavní tok: 1. Systém zobrazí seznam uložených konfigurací. 2. *Uživatel* vybere některou konfiguraci. 3. Případ použití se spustí, když *Uživatel* vybere položku „Načíst konfiguraci“. 4. Systém načte vybranou konfiguraci. 5. Systém zobrazí aktualizovaný obsah nástroje podle konfigurace.
Následné podmínky: Žádné.
Alternativní toky: Žádné

Případ použití: Mazání konfigurace
ID: PP6
Stručný popis: Odebrání konfigurace ze systému ladícího nástroje.
Účastníci: Uživatel
Vstupní podmínky: Autentizovaný *Uživatel*.
Hlavní tok: 1. Systém nejprve zobrazí přehled všech dostupných konfigurací systému. 2. Případ použití se spustí, když *Pokročilý uživatel* zvolí „Odebrat konfiguraci“. 3. Systém odebere vybranou konfiguraci. 4. Systém zobrazí aktualizovaný seznam konfigurací.
Následné podmínky: Žádné.
Alternativní toky: Žádné

Příloha B

Popis instalace nástroje

Požadavky na chod nástroje:

Nástroj ke svému fungování vyžaduje webový server, který podporuje skriptovací jazyk PHP verze 5 a vyšší, standardně Apache.

Popis zprovoznění:

Popis zprovoznění nástroje s dodanou ukázkovou aplikací

1. Prvním krokem je zvolení vhodného webového serveru.
2. Druhým krokem je nakopírování obsahu adresáře `EXAMPLE_APP` z přiloženého CD do kořenového adresáře určeného pro webové stránky na serveru.
3. Následně je nutné přidat svou IP adresu, pod kterou budeme přistupovat ke stránkám na serveru, do určených míst souboru na serveru s cestou **`EasyDebug/settings/permission.php`**. Určená místa jsou:
   - Oblast mezi „`// START`" a "`// END`" – IP adresy v této oblasti mají veškerá oprávnění nástroje kromě přidávání a mazání existujících uživatelů
   - Oblast mezi „`// SADMIN`" a „`// EADMIN`" – IP adresy v této oblasti mají veškerá oprávnění nástroje včetně přidávání a mazání exitujících uživatelů

   Následné přidávání/odebírání IP adres (tj. uživatelů) již lze provádět přímo z nástroje
4. První tři kroky by měly stačit ke zdárnému zprovoznění aplikace spolu s ladícím nástrojem. Pokud tomu tak přesto není a ladící nástroj se nezobrazuje, nebo se zobrazuje s poškozeným vzhledem ve spodní části stránky, nejpravděpodobněji je nutné nastavit v souboru **`EasyDebug/init.php`** konstantu `EDEBUD_DIR` na cestu od kořene webu po adresář `EasyDebug`, jako např. „`/Easydebug`". Tato komplikace bývá způsobena rozdílným nastavením různých serverů.

Popis zprovoznění s vlastní aplikací

1. Prvním krokem je zvolení vhodného webového serveru.
2. Dále je třeba nakopírovat obsah adresáře `SOURCE` z přiloženého CD do kořenového adresáře aplikace.
3. Aby byl nástroj začleněn do aplikace, je třeba přidat na začátek a konec stránky s aplikací, kterou chceme ladit, řádky:

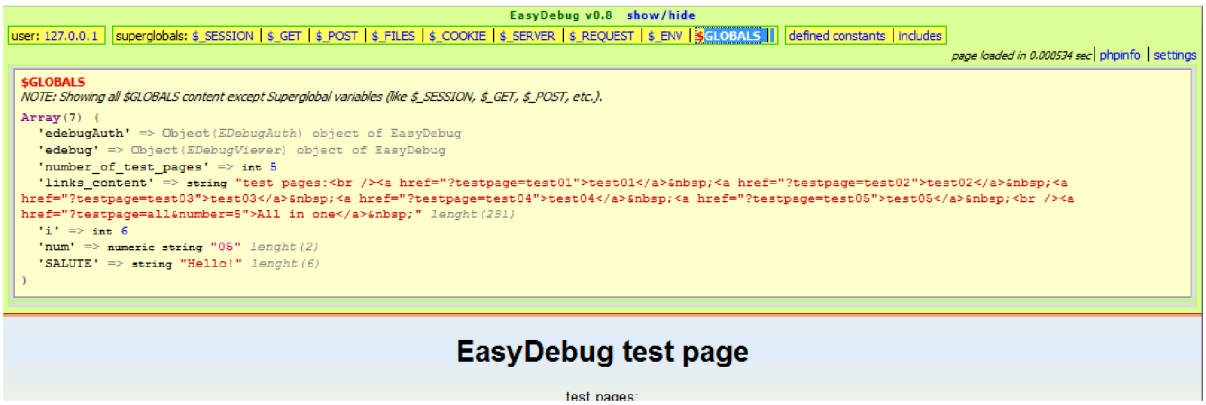
- Na začátek: `<?php require_once 'EasyDebug/init.php'; ?>`
- Na konec: `<?php $edebug->pageEnd(); ?>`

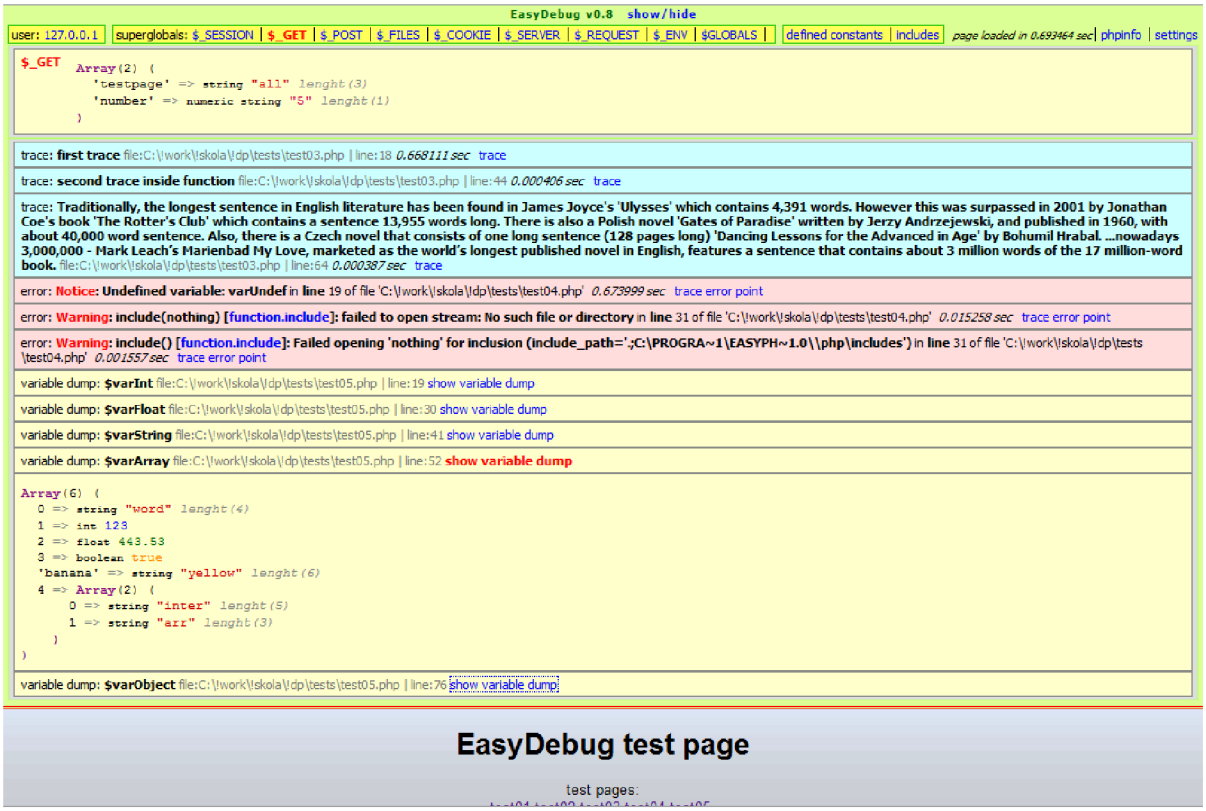
POZN. Pokud je požadováno, aby byl HTML kód aplikace validní i při používaní nástroje, je nutné umístit řádek `<?php $edebug->pageEnd(); ?>` těsně před element `</body>`, který uzavírá tělo HTML.

4. Dále lze pokračovat bodem 3 předešlého popisu zprovoznění nástroje s dodanou ukázkovou aplikací.

Příloha C

Ukázka vzhledu výsledného nástroje





Příloha D

Obsah přiloženého CD

`DOCUMENTATION`	- Adresář, který obsahuje dokumentaci aplikačního programátorského rozhraní nástroje.
`EXAMPLE_APP`	- Adresář, který obsahuje zdrojové kódy ukázkové aplikace znázorňující příklady použití nástroje. Obsahuje rovněž všechny zdrojové soubory nástroje.
`MANUAL`	- Adresář, který obsahuje uživatelskou příručku pro používání nástroje.
`SOURCE`	- Adresář, který obsahuje všechny zdrojové kódy a potřebné soubory nástroje.
`TEXT`	- Adresář, který obsahuje technickou zprávu této diplomové práce ve formátu .pdf.
`TEXT-SOURCE`	- Adresář, který obsahuje zdrojové kódy a soubory technické zprávy této diplomové práce.
`readme.txt`	- Textový soubor, který popisuje obsah CD