TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

DIPLOMOVÝ PROJEKT

Liberec 2013 **Ondřej Smola**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Návrh a implementace architektury pro tvorbu single page web aplikací

Design and implementation of architecture for creating single page web applications

Diplomový projekt

Autor: **Bc. Ondřej Smola**

Vedoucí práce: Ing. Jan Silovský, Ph.D.

V Liberci dne 17.5.2013

zadání

Prohlášení

Byl jsem seznámen s tím, že na můj diplomový projekt se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Na tomto místě bych rád poděkoval Ing. Janu Silovskému, Ph.D., za odborné vedení diplomové práce a cenné připomínky při jejím sepisování.

Abstrakt

Cílem této práce je návrh a implementace aplikačního rámce pro tvorbu single page aplikací. Nejprve je popsán použitý programovací jazyk a jeho možnosti pro potřeby tvorby aplikačního rámce. Následně je stručně popsána problematika a přínos single page aplikací a diskutovány hlavní výhody a nevýhody zvoleného přístupu k tvorbě plně interaktivních webových stránek. Hlavní částí práce je návrh. Rámec je nejprve rozdělen do hlavních komponent a je popsána celková architektura aplikačního rámce s umístěním jednotlivých komponent. U každé komponenty je stručně diskutována její funkcionalita a graficky znázorněno její umístění uvnitř celého rámce. Následuje implementační část, která popisuje způsob implementace hlavních logických celků aplikačního rámce společně s diskuzí problémů vycházejících ze zvoleného způsobu implementace. Na závěr práce jsou popsány komponenty vytvořené nad rámec zadání a zhodnoceno celkové řešení.

Klíčová slova: SPI, JavaScript, aplikační rámec, dynamické webové stránky

Abstract

The aim of this work is the design and implementation of application framework for creating single-page web applications. At first, used programming language is described together with its support for application framework creation. Single page application problems and benefits are then briefly described and the main advantages and disadvantages of the approach to create a fully interactive website are discussed. The main part of this work is design. The framework is divided into main components and the overall architecture of an application framework together with the place of each component is described. Each component functionality is briefly discussed and its location with respect to the context of the framework is graphically illustrated. Implementation section describes how the main logic units of application framework are implemented, together with a discussion of problems arising from implementation of the chosen method. The last part describes the components created beyond the specification and evaluates the overall solution.

Keywords: SPI, JavaScript, application framework, dynamic web pages

Obsah

Seznam ilustrací

Úvod

Poslední desetiletí znamenalo z hlediska vývoje webových technologií obrovský skok kupředu. Webové stránky se postupně z klasických statických dokumentů stávají místem dynamické interakce s uživatelem. V posledních několika letech se ovšem vývoj s ohledem na úroveň možností webových stránek prakticky zastavil. Nové technologie pro tvorbu webových aplikací vznikají každým dnem, ale z pohledu uživatele nedochází ve většině případů k posunu interaktivity ani na úroveň základních aplikací, které zná z prostředí pracovní plochy svého počítače. Problém je dále zveličován masivní rozšířením a popularizací on-line bankovnictví, emailových klientů nebo stránek pro vzájemnou interakce mezi lidmi, ať už se jedná chat či nástroje pro spolupráci mezi spolupracovníky. Uvedené typy webových aplikací již z principu potřebují velkou část funkcionality realizovat dynamicky a proto poslední dobou dochází k značnému rozvoji technologií usnadňujících tvorbu interaktivních aplikací.

Mezi jedno z často diskutovaných témat z oblasti budoucnosti webových stránek je realizace plně dynamické aplikace, označované jako single page aplikace (SPI). Cílem SPI je poskytnout uživateli komfort desktopových aplikací uvnitř webového prohlížeče. Mezi stávající úspěšné implementace patří některé portály bank či služby poskytované velkými společnosti, jakými jsou například společnosti Google, Microsoft či LinkedIn.

Tvorba SPI aplikací je ale spojena s celou řadou problémů a její vývoj je v porovnání s klasickým vývojem webové aplikace jiný a v určitých ohledech i mnohem náročnější, jak na schopnosti, tak zkušenosti vývojového týmu. Během posledních měsíců lze zaznamenat velký rozvoj aplikačních rámců a knihoven, které vytváří abstrakci pro podporu tvorby SPI. Jejich zaměření je ale jen na určitou část celé problematiky zatímco cílem této práce je vytvořit aplikační rámec, který bude vytvářet ucelený způsob tvorby SPI aplikací.

1 Analýza problematiky tvorby SPI

1.1 Javascript – jazyk pro tvorbu SPI

Jedná se o multiplatformní objektově orientovaný skriptovací jazyk. Hlavním důvodem jeho vzniku byla snaha vytvořit jazyk umožňující řízení logiky uvnitř webového prohlížeče a interakci s objektovým modelem webového dokumentu. Jazyk Javascript[1] byl původně vyvíjen pod názvem LiveScript a následné slovní spojení s jazykem Java bylo pouze snahou získat lepší postavení na trhu, jelikož se jazyk Java v této době stával velice populární. Jedná se o aktuálně jeden z nejrozšířenějších a nejpoužívanějších programovacích jazyků. Za jeho rozšířením stojí především masivní použití v moderních interaktivních webových aplikacích a v posledních několika letech i jeho využití na straně serveru.

1.2 Javascript a funkcionální programování

Mezi jednu z hlavních pozitiv jazyka Javascript patří podpora pro některé konstrukty z funkcionálním programování. Mezi základní konstrukty patří funkce první třídy, což zjednodušeně znamená, že s instancemi funkcí lze manipulovat jako s instancemi jiných objektů, předávat je jako parametry funkcí, definovat metody nad nimi a přiřazovat je do proměnných. Funkce si uchovávají pohled na prostředí, ve kterém byly vytvořeny. Pohled na prostředí v době vytvoření funkce společně s danou funkcí se obvykle označuje jako *closure*. C*losure* je jedním z hlavních způsobů jak realizovat zapouzdření přístupu k vnitřním (instančním) proměnným. Jazyk Javascript podporuje metody pro zapouzdření přístupu k vnitřním proměnným až od standardu ECMAScript verze 5.1[2], který stále nelze považovat za zcela rozšířený. Mezi další konstrukty patří podpora pro lambda výrazy, běžně nazývané jako anonymní funkce.

I bez podpory celé další řady konstruktů, jakými jsou například částečná aplikace funkce [3], podpora optimalizace pro *tail-rekurzi* nebo podpora pro předávání argumentu funkcí jménem (*call by name*), umožňuje Javascript velkou část logiky realizovat funkcionálně. Jelikož samotný pojem funkcionálního programování si lze vyložit několika způsoby a jedním z nich je podpora pro funkce první třídy a lambda výrazy, lze jazyk Javascript považovat kromě objektově orientovaného také za funkcionální.

1.3 Single page aplikace

Mezi základní důvody vzniku tohoto termínu, který nebyl doposud přesně definován, je snaha popsat webové aplikace, které poskytují uživateli komfort a některé možnosti desktopové aplikace na úrovni webové stránky. Zjednodušeně lze SPI definovat jako webovou aplikaci, na které většina změn probíhá v aktuálním dokumentu a pro zobrazení informací, či reakci na akci uživatele, je reagováno asynchronně. Klasickým rozdílem mezi dynamickým webem a SPI je navigace, kdy u dynamického webu dochází k načtení všech, z velké části redundantních, dat. SPI si klade za cíl načíst nejmenší možné množství informací tak, aby bylo možné plně obsloužit uživatelský požadavek. Chování SPI lze uvést na způsobu chování aplikací pro správu souborů, kdy otevření složky znamená pouze zobrazení složek a souborů a nedochází ke znovu načtení ovládacích prvků a ostatních nesouvisejících panelů, tak jak by tomu bylo v případě klasické webové aplikace.

Mezi hlavní výhody stránek implementovaných podle vzoru SPI je především rychlost a interaktivita. Interaktivita spočívá především v prakticky neomezené volnosti v logice na straně klienta. Z tohoto důvodu je návrh SPI velice oblíbený při implementaci emailových klientů, či aplikací vyžadující spolupráci několika lidí. S interaktivitou souvisí i možnost jednoduše pracovat s daty a pohledy vytvořenými během celého životního cyklu aplikace, jelikož data nejsou ztracena při přechodu na jinou stránku. Mezi hlavní výhodu SPI patří také úspornost. Jelikož nejsou přenášena redundantní data, dochází k značnému snížení celkových nároků na alokovanou přenosovou kapacitu internetové sítě a dochází ke snížení výpočetních nároků na cílový webový server i klienta. Úspornost je výhodná především u mobilních zařízení, které obvykle disponují řádově pomalejším internetovým připojením a obvykle jsou navíc značně omezeny množstvím přenesených dat.

Mezi hlavní nevýhody patří samotný jazyk JavaScript, který nebyl původně určen pro tvorbu SPI aplikací a proto je třeba při vývoji dbát mnoha pravidel, aby vzniklá implementace byla rozšiřitelná a její údržba se nestala noční můrou. Mezi zcela nejkritičtější nedostatky jazyka JavaScript patří nulová podpora pro dělení aplikace do funkčních bloků a sdílení globálního prostoru všemi skripty. Další nevýhodou je samotný koncept webu jako propojených dokumentů a s tím související historie navigace a tlačítko zpět, které není jednoduché v aplikaci SPI správně implementovat. Poslední nevýhodou je nutnost vyvíjet webovou aplikaci jako SPI od začátku, jelikož přechod v pozdějším stádiu projektu je velice obtížný.

1.4 Dostupná řešení

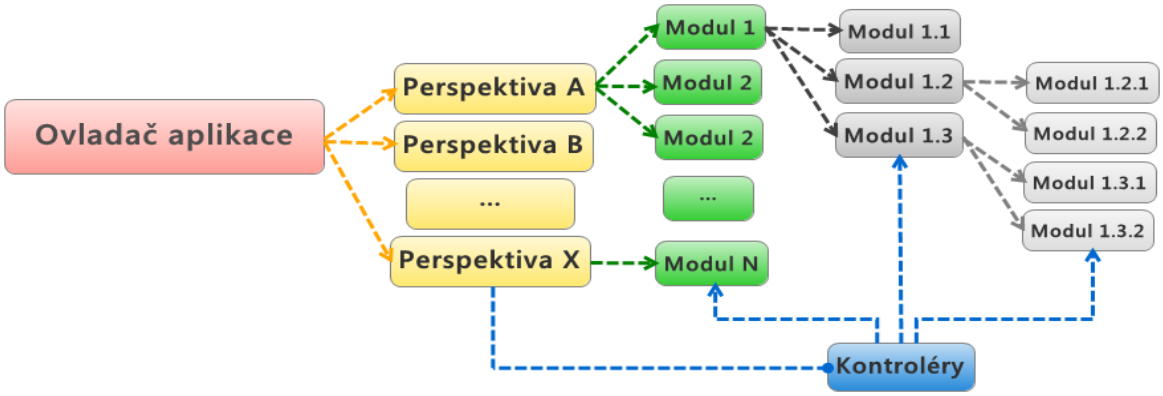
Aktuálně neexistuje ucelený aplikační rámec, který by poskytoval plnou podporu pro tvorbu SPI aplikací. Většina stávajících řešení řeší pouze oddělené problémy a je nutné je kombinovat či upravovat pro jednotlivé potřeby aplikací. Mezi jedny z rozšířených aplikačních rámců, které řeší některou z částí dané problematiky patří Backbone.js[4], jQuery, Ember a Angular.js[5].

2 Návrh aplikačního rámce

Mezi hlavní cíle aplikačního rámce patří poskytování infrastruktury pro tvorbu SPI aplikací. Infrastruktura se bude skládat z následujících hlavních částí:

- Popis způsobu definice a implementace modulu
- Způsob načítání jednotlivých modulů a jejich závislostí
- Řešení vzájemné komunikace mezi moduly
- Podpůrné implementace

Jelikož existuje již existuje několik implementací, které řeší některé z výše uvedených problémů, bude během práce kladen důraz na vyřešení dané problematiky uceleným způsobem, tak aby nebylo třeba spojovat stávající řešení. Architekturu rámce lze při pohledu shora rozdělit na ovladač aplikace, perspektivy moduly a kontroléry (Obr. 1).



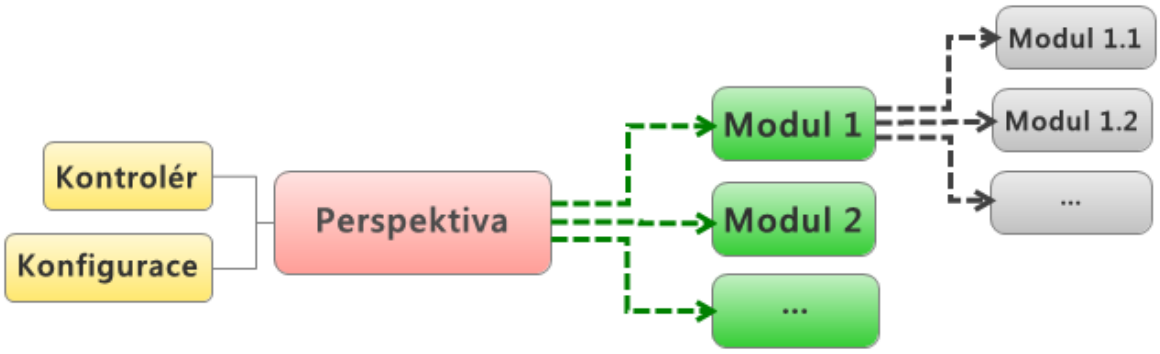
Obr. 1: Celkový pohled na architekturu rámce z hlediska jednotlivých komponent

2.1 Ovladač aplikace

Obr. 2: Návrh ovladače aplikace se zobrazením základního API pro uživatele

Ovladač aplikace (Obr. 2) je jediné místo přímé interakce programátora s aplikačním rámcem. Hlavním a prakticky jediným úkolem ovladače je spuštění rekurzivního algoritmu pro načtení všech modulů. Ovladač dále obsahuje vlastní globální kontrolér, do kterého jsou registrovány události potřebné pro celkový chod aplikace, označované jako globální události.

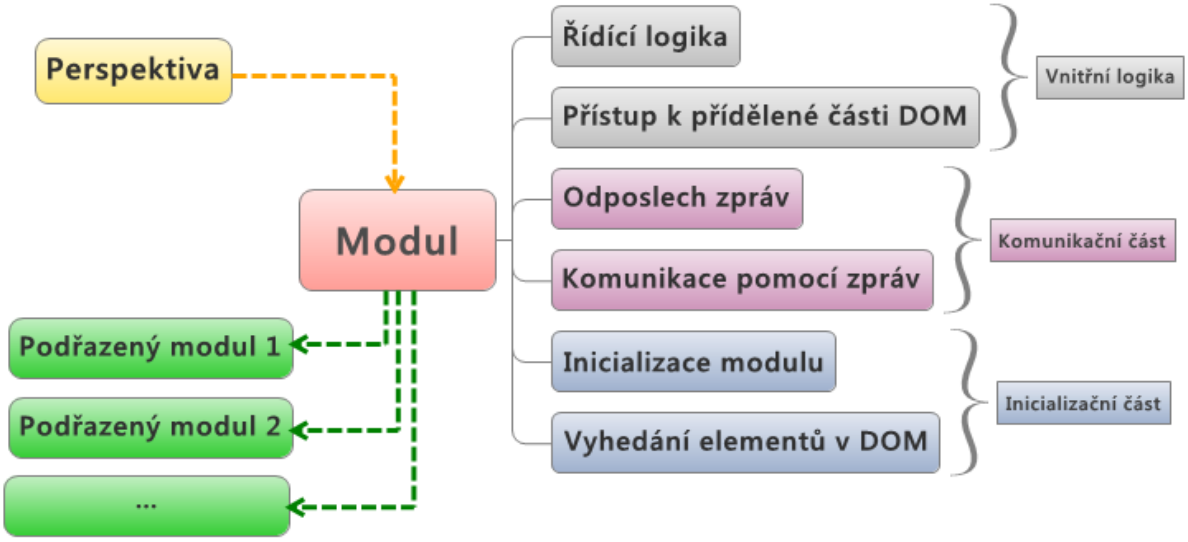
2.2 Perspektiva



Obr. 3: Umístění perspektivy uvnitř rámce se zobrazením stromu modulů

Hlavním prvkem webové stránky je perspektiva (Obr. 3). Perspektiva utváří pohled na webovou stránku a slouží pro reprezentaci skupiny modulů. Perspektivy nahrazují změnou cílové perspektivy přechod mezi jednotlivými webovými stránkami u běžné webové aplikace. Perspektiva obvykle neobsahuje žádnou vnitřní logiku a představuje kořenový element stromu modulů. V aktuálním nastavení navíc slouží pro definici hlavního kontroléru, který je poté možné sdílet pro všechny moduly obsažené uvnitř perspektivy.

2.3 Modul

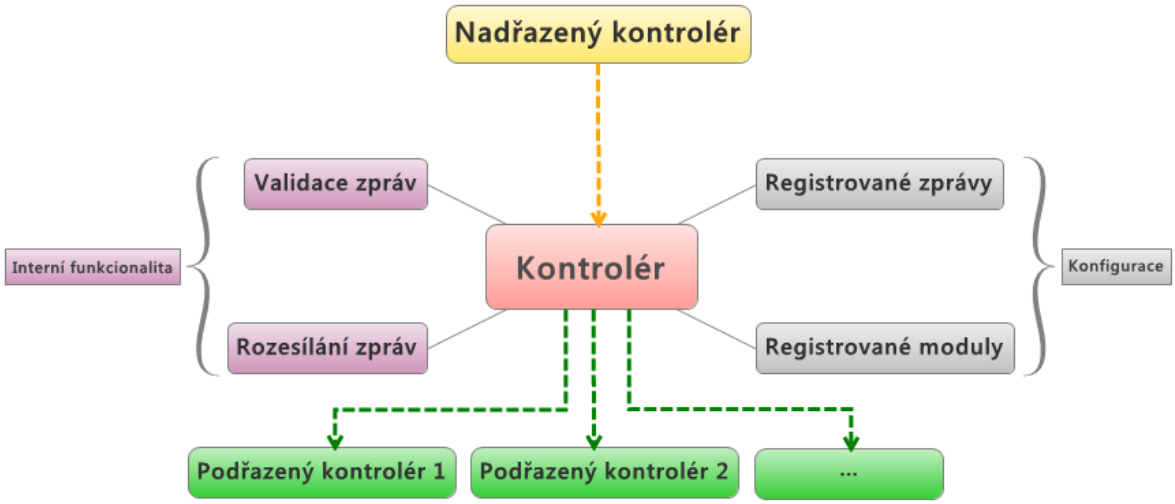


Obr. 4: Hierarchické zobrazení modulu uvnitř rámce s popisem jeho hlavních částí

Základním stavebním prvkem perspektivy je modul. Modul bude definovat vnitřní funkcionalitu svázanou s části objektovým modelem dokumentu (DOM [6]). Mezi klíčové vlastnosti bude patřit naprosté oddělení od okolního prostředí a z toho vyplývající nemožnost komunikovat s ostatními moduly přímo. Během inicializace bude modulu přidělen kontrolér

a identifikační objekt. Identifikačním objektem je objekt obsahující informace pro inicializaci modulu, případně uživatelem definované atributy. Jedinou povolenou cestou komunikace s okolním prostředím je posílání zpráv pomocí přiděleného kontroléru a odposlech zpráv jím rozesílaných.

2.4 Kontrolér

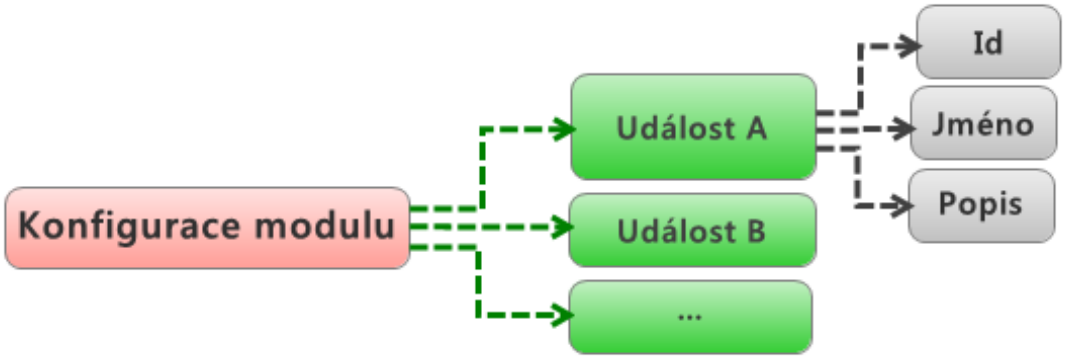


Obr. 5: Hierarchické umístění kontroléru společně s popisem jeho funkcionality

Kontroléry (Obr. 5) budou mít na starosti správu komunikace mezi jednotlivými moduly. Na základě konfigurace zjistí kontrolér přiřazené události a moduly. Na základě konfigurace je poté pokus o odeslání zprávy z určitého modulu povolen či zamítnut vyvoláním výjimky. Kontrolér jako jediný v celém systému bude mít k dispozici informace o tom, které moduly jsou v aplikaci přítomny a které události mohou používat. Pokud modul bude chtít odeslat zprávu ostatním modulům, musí použít jemu přidělený kontrolér. Modul se bude moci identifikovat pomocí svého identifikátoru tak, aby mohl vyloučit sebe z cílových modulů pro odeslání. Tento způsob umožní oboustrannou komunikaci pomocí jediné zprávy tak, aby nedošlo k odposlechu vlastní zprávy, což by následně mohlo uvést odesílání zpráv do nekonečného cyklu.

Kontroléry budou uspořádány do stromové struktury. Kořenem této struktury bude kontrolér perspektiv, kterému budou podřazeny kontroléry jednotlivých perspektiv. Kontroléry perspektiv mohou být sdíleny mezi moduly, nebo bude možné definovat kontroléry i pro jednotlivé moduly či rekurzivně dále i pro podřazené moduly. Cílem rekurzivní struktury je umožnění jak globálního, tak lokálního odesílání zpráv. Jako příklad globální zprávy lze uvést požadavek o změnu aktuální perspektivy a příkladem lokální zprávy je příkaz pro zobrazení načteních dat.

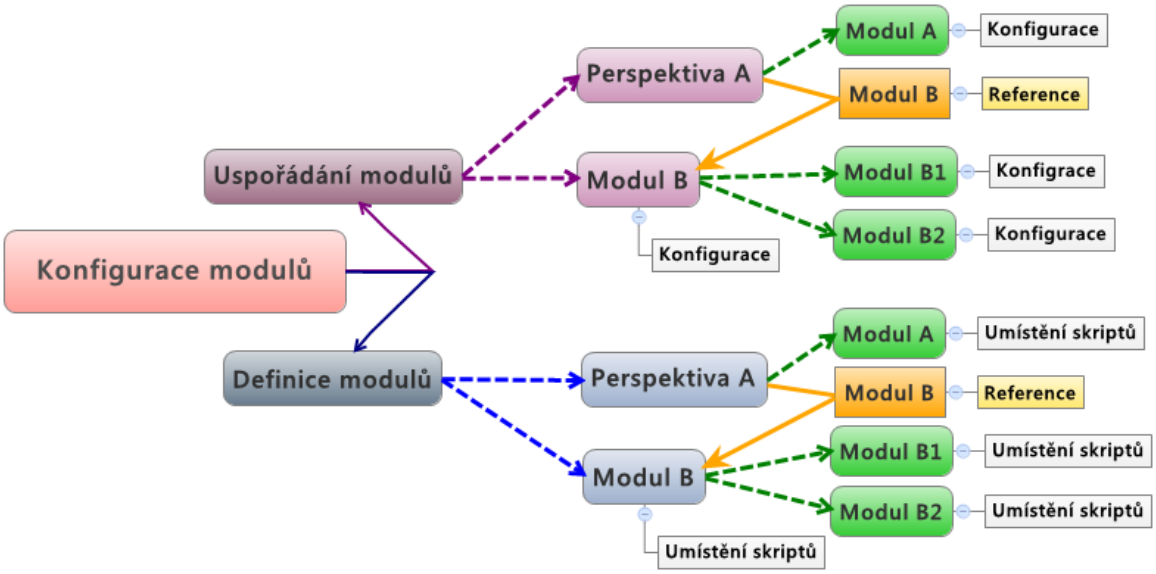
2.5 Událost



Obr. 6: Uspořádání událostí uvnitř rámce

Úkolem událostí bude zapouzdřit a popsat změnu stavu v aplikaci. Událost (Obr. 6) obsahuje slovní popis, který je použit pro vyhledávání a identifikaci události v programu. Název události musí být jedinečný pouze v rámci jedné úrovně, ale nemusí být jedinečný v rámci celé aplikace. Z důvodu jednoznačné identifikace události v systému je jí během inicializace přidělen unikátní identifikátor. Součástí události je i volitelný popis, který slouží k poskytnutí informací o očekávané funkci a případně i o datech odesílaných společně s událostí.

2.6 Konfigurace modulů



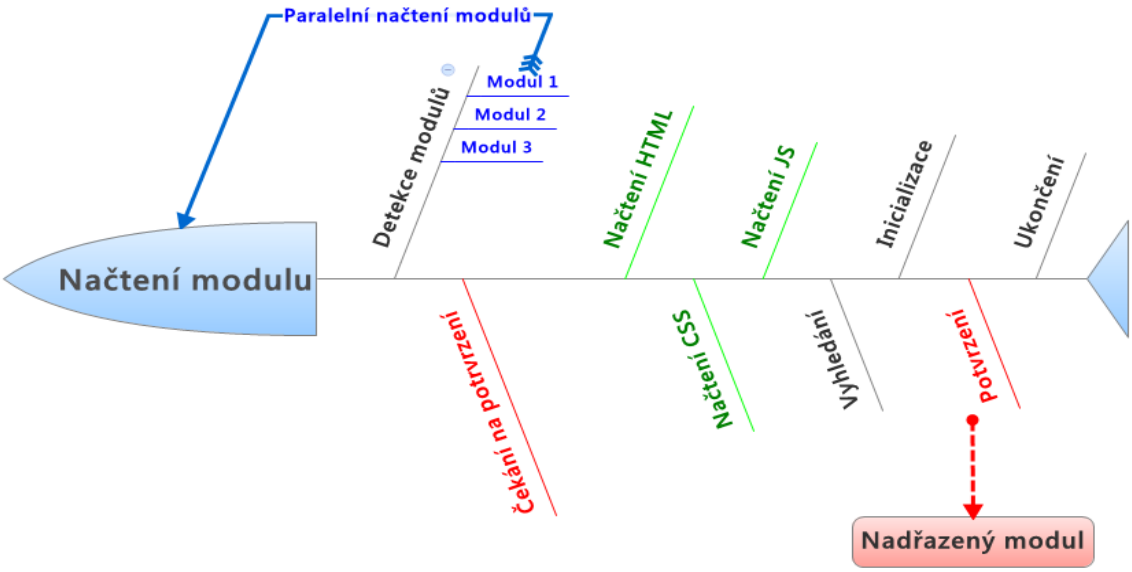
Obr. 7: Stromové uspořádání použité pro definici modulů

Konfigurace modulu (Obr. 7) je rozdělena do části pro popis struktury, která zároveň obsahuje konfiguraci pro daný modul, a definiční oblast obsahující informace o umístění jednotlivých zdrojů. V případě, že modul obsahuje další podřazené moduly, musí být jeho definice umístěna vedle perspektiv a v samotném stromu je na něj pouze vytvořena reference.

3 Implementace

Implementaci lze rozdělit do třech hlavních částí. Základními částmi je způsob načtení a realizace modulů, dále pak implementace logiky rozesílání událostí a implementace podpůrných komponenty.

3.1 Načtení modulu

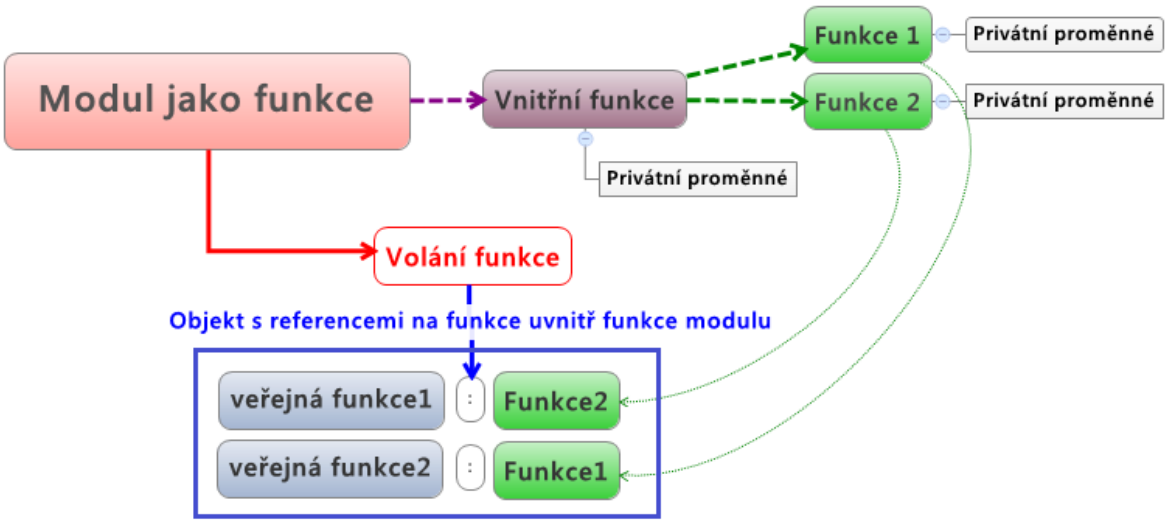


Obr. 8: Popis jednotlivých kroků vytvoření modulu

Načítání modulů (Obr. 8) je vyvoláno přes ovladač aplikace. Pomocí předaného identifikátoru perspektivy je v souboru s konfigurací vyhledán objekt s identickým identifikátor. Na základě konfigurace objektu reprezentujícího perspektivu je v případě, že perspektiva neobsahuje žádné moduly, načtena okamžitě, nebo jsou nejprve rekurzivně načteny všechny podřazené moduly. Během rekurzivního volání jsou modulům předávány kontroléry nadřazených modulů a postupně načítané konfigurační objekty. Rekurzivní volání je koncipováno jako *zdola-nahoru,* neboli výsledek se začíná skládat až po detekci všech modulů v podstromu stromu modulů. Díky využití možností funkcionálního programování je tento celkem komplikovaný návrh možný realizovat poměrně jednoduše kompozicí funkcí a předávání referencí na funkce. Princip lze jednoduše popsat tak, že každý modul předá podřazeným modulům referenci na funkci, kterou zavolají po asynchronním načtení všech svým závislostí. Poté co všechny podřazené moduly zavolají předanou funkci je možné načíst nadřazený modul, který po načtení zavolá jemu předanou funkci. Rekurzivní volání končí

zavoláním funkce kořenového modulu, který poté dokončí inicializaci perspektivy. Načtení samotného modulu se skládá z volitelného načtení fragmentu HTML, dále pak z načtení kaskádových stylů a skriptů v jazyce JavaScript. Po načtení volitelných částí je vyhledán objekt reprezentující modul na globálním objektu a je spuštěna jeho inicializace.

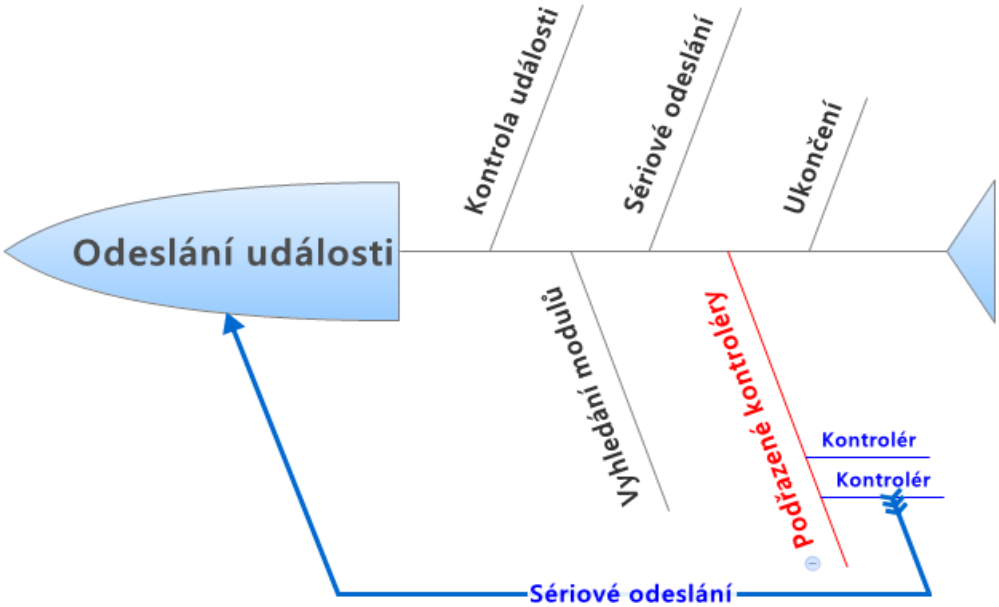
3.2 Implementace modulu



Obr. 9: Vytvoření objektu reprezentujícího modul pomocí funkce

Z důvodů zapouzdření vnitřních stavů modulů a z důvodu prevence neúmyslnému zanášení globálního prostředí aplikace musí být modul realizován jako funkce, která při své vytvoření zavolá sama sebe. Výsledkem tohoto volání musí být objekt (Obr. 9), který jako jediné atributy obsahuje reference na *closure* definované uvnitř samotné funkce. Tato realizace umožňuje definovat privátní proměnné tak, aby nebyly přístupné z globálního prostoru

3.3 Odeslání události

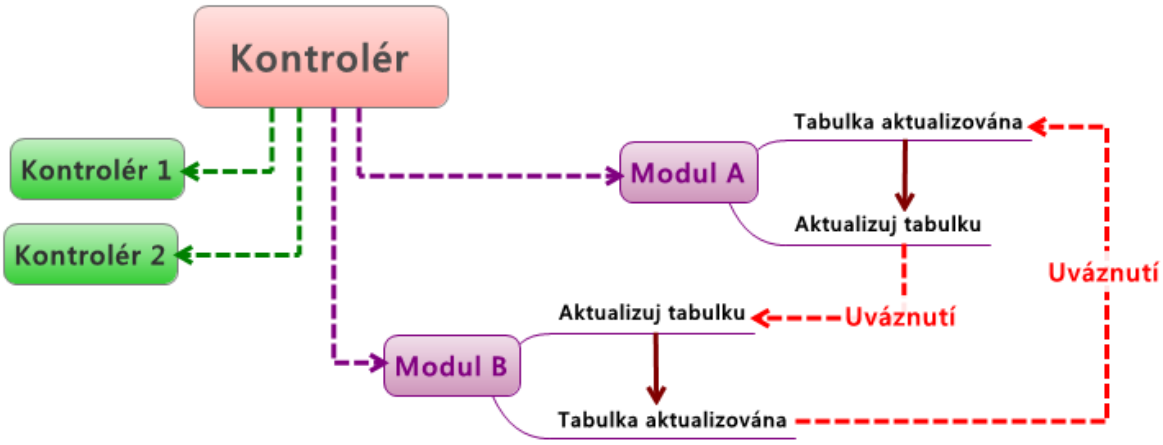


Obr. 10: Jednotlivé kroky při odeslání události

Odeslání události (Obr. 10) patří z pohledu aplikačního rámce k nejčastější události. Proto bylo potřeba implementovat řízení událostí efektivním ale zároveň i robustním způsobem. Událost je v aplikace reprezentována pojmenovanou zprávou s volitelnými daty. Každý modul při své inicializaci obdrží kontrolér, který jediný smí používat pro odesílání zpráv okolním modulů.

Při pokusu o odeslání události je nejprve pomocí kontroléru vyhledána požadovaná zpráva v konfiguračním souboru. Vyhledáním zprávy je umožněna, jak kontrola oprávnění, tak detekce neznámého názvu zprávy. Po vyhledání je zpráva pomocí kontroléru odeslána. Kontrolér na základě známých modulů vyhledá všechny cíle a zprávu začne sériově odesílat jednotlivým modulům. V případě, že zprávu obdrží jiný kontrolér, dojde k rozeslání zprávy všem modulům registrovaným u tohoto kontroléru. Tento postup je realizován jako *zhora-dolů* a je tedy rekurzivně opakován do doby, než dojde k odeslání zprávy poslednímu modulu. Odeslání události je obvykle následováno odesláním dalších zpráv z jiných modulů a následně dalších jako reakce na ně. Z tohoto důvodu je nutné, aby obsluha událostí uvnitř jednotlivých modulů netrvala příliš dlouho, nebo byla provedena asynchronně.

3.3.1 Uváznutí při odesílání událostí

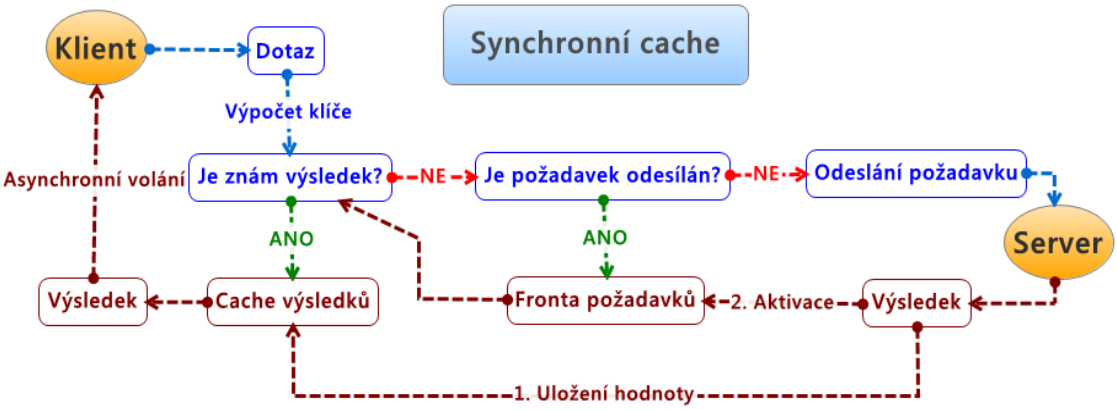


Obr. 11: Grafické zobrazení problematiky uváznutí během odesílá událostí

Díky uspořádání kontroléru do stromové struktury není možný vznik nekonečných cyklů na úrovni hierarchie rozesílání zpráv. Hlavním a prakticky jediným možným místem vzniku nekonečného cyklu je v případě, kdy alespoň dvě funkce obsluhující události křížově vyvolávají sami sebe (Obr. 11). Ke vzniku uváznutí může dojít také v případě, že modul vyvolá při obsluze události událost znovu. Pro tento případ obsahuje kontrolér funkci pro odeslání všem, kromě předaného modulu. V případě odesílání pouze ostatním modulům je nejprve podle poskytnutých identifikátorů vyhledány moduly, které jsou poté vyloučeny ze seznamu cílových modulů.

3.4 Podpůrné implementace

3.4.1 Synchronní cache



Obr. 12: Realizace asynchronní cache pomocí stavového automatu

Cache pracuje na principu, který lze popsat stavovým automatem (Obr. 12). Nejprve je vypočítán identifikátor dotazu na základě předané hash funkce. Na základě navrácené hodnoty je poté dotaz obsloužen přímo z cache nebo předán dále ke zpracování. Tímto by končila funkcionalita klasické implementace cache. Implementovaná cache dále synchronizuje dotazy tak, aby nedocházelo k několika (z hlediska hash funkce) identickým asynchronním dotazům. Každý dotaz, který není v cache, nastaví před samotným odesláním požadavku příznak cache tak, aby následující požadavky se stejným hashem byly místo odeslání uloženy do fronty. Poté co je získán výsledek požadavku, je hodnota uložena do výsledků a je aktivována fronta čekajících požadavků se stejným hashem, ty jsou navráceny do stavu kdy kontrolují, zda-li již není znám výsledek, který v tomto případě již známý je.

3.4.2 Komponenta pro obsluhu asynchronní komunikace

Hlavním úkolem komponenty je tvorba centralizované správy komunikace se serverem. Komponenta je definována podobně jako modul a výsledkem jejího vytvoření je objekt obsahující reference na poskytované funkce. Pro komunikaci je použita definovaná funkce s volitelnými argumenty. Mezi volitelné argumenty patří data požadavku a reference na funkce po úspěšném a neúspěšném dotazu. Komponenta zároveň obsluhuje neúspěšné volání a zobrazí chybové hlášení buď do konzole webového prohlížeče, nebo zobrazí jako dialogové okno uživateli. Komponenta slouží zároveň i pro dokumentaci rozhraní mezi aplikací a webovým serverem.

Závěr

Cílem práce bylo navrhnout a vytvořit aplikační rámec pro tvorbu SPI aplikací. Nejprve byla provedena analýza stávajících řešení. Na základě analýzy nebylo nalezeno žádné volně dostupné řešení, které by poskytovalo podporu pro většinu problémů spojených s tvorbou SPI. Proto bylo rozhodnuto vytvořit zcela vlastní návrh, zaměřující se na rekurzivní realizaci logiky.

Základními prvky rámce jsou moduly, události, kontroléry a perspektivy. Zvolený návrh se odlišuje od ostatních řešení především v tom, že realizuje většinu logiku zcela rekurzivně a dbá na podporu hierarchií a závislostí, čímž lépe odpovídá tomu, jak jsou moderní webové stránky tvořeny. Během návrhu byla vyřešena jak logika rekurzivního načítání jednotlivých komponent, tak i způsob vzájemné synchronizace tak,a to způsobem aby nedocházelo k neočekávanému chování. Dále byl definován a vytvořen způsob hierarchické komunikace mezi moduly včetně ošetření nekonečných cyklů.

Na základě návrhu byla poté provedena implementace. Během implementace byly navíc vytvořeny komponenty pro tvorbu synchronních cache a komponenta pro podporu asynchronních volání s automatickou obsluhou chybových stavů. Aplikační rámec je aktuálně používán pro tvorbu aplikace vyvíjené na Technické Univerzitě v Liberci. Jelikož tato aplikace není aktuálně veřejně dostupná, byla pro demonstraci vytvořena webová stránka, která zároveň slouží jako základní dokumentace.

Demonstrované řešení je přizpůsobeno pro potřeby vyvíjené aplikace a je možné jej jednoduchými úpravami upravit pro potřeby jiných aplikací. Dalšími kroky během vývoje rámce bude vyřešení problematiky historie při navigaci na SPI stránce a implementace komponenty pro podporu změny jazykového nastavení.

Seznam použité literatury

[1] Javascript. *MOZILLA DEVELOPER NETWORK* [online]. 2013 [cit. 2013-04-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/JavaScript

[2] ECMAScript® Language Specification. ECMA. *Standart ECMA-262* [online]. 2011 [cit. 2013-04-29]. Dostupné z: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[3] Currying. [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://www.haskell.org/haskellwiki/Currying

[4] BackboneJS. [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://backbonejs.org/

[5] AngularJS. *MOZILLA DEVELOPER NETWORK* [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://angularjs.org/

[6] Document Object Model. [online]. 2013 [cit. 2013-04-29]. Dostupné z: http://www.w3.org/DOM/

[7] O'REILLY MEDIA / YAHOO PRESS. JavaScript: The Good Parts [online]. 2008. ISBN978-0-596-15873-6. Dostupné z: http://shop.oreilly.com/product/9780596517748.do

[8] O'REILLY MEDIA. JavaScript Patterns [online]. 2010. ISBN 978-0-596-80677-4. Dostupné z: http://shop.oreilly.com/product/9780596806767.do

Příloha A

Demonstraci vytvořeného řešení si lze prohlédnout na:

http://cybersearch.ite.tul.cz/sp