Podpora implementácie GWT klienta

BAKALÁRSKA PRÁCA

Milan Mandák

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

9.2.1 Informatika

Vedúci záverečnej práce
Mgr. Martin Jaška

BRATISLAVA 2010

Týmto prehlasujem, že som bakalársku prácu vypracoval samostatne s odbornou pomocou školiteľa a s použitím uvedenej literatúry.

Bratislava, jún 2010 Milan Mandák

Abstrakt

Práca sa zaoberá podporou implementácie GWT klienta. Samotné GWT totiž nemá podporu pre prenos dát, validáciu a ďalších súčastí, ktoré sú nevyhnutné na implementáciu ktorejkoľvek web aplikácie.

Cieľom práce je pokryť už spomenuté oblasti nevyhnutné pre implementáciu web aplikácie. Teda naprogramovať funkčný framework a zároveň aj Demo aplikáciu na predvedenie frameworku.

Obsah

1. Úvod

Web aplikácie napísané pomocou AJAX-u sú možno budúcnosťou moderného webu avšak sa ťažšie vyvíjajú. Preto ma zaujal framework GWT, ktorý vytvára oveľa pohodlnejšie prostredie na implementáciu, hľadanie chýb a aj udržiavanie kódu väčších projektov. Avšak GWT je nástroj iba na vystavanie užívateľského rozhrania a teda chýba prepojenie so serverom, ktorý by mal na starosti ukladanie dát do databázy, validáciu dát pozbieraných z GUI a rôznu ďalšiu logiku. Preto som sa rozhodol pokúsiť spraviť takýto podporný framework. Je jasné, že rozsah takéhoto projektu môže narásť do veľkých rozmerov kvôli množstvu rôznych widgetov implementácii ich použitia. Preto sa zameriam na kostru projektu. Čo to presne znamená? Presne to znamená, že sa budem sústrediť na implementáciu hlavnej logiky ako je návrh dát, ktoré sa budú prenášať medzi klientom a serverom pri validácii, aktualizácií widgetov, aktualizácií dát na servery, spúštaní logiky na servery.

Výsledkom celej práce by mal byť funkčný framework s limitovanou funkčnosťou, ktorá sa ale bude dať jednoducho doplniť

2. Java Enterprise Edition

2.1. História

Java enterprise edition je pomerne nová technológia, ktorá uzrela svetla sveta v roku 1999 keď bol vydaný prvý beta software. Ďalším míľnikom pre J2EE je rok 2001 keď počet stiahnutí J2EE SDK(Java Enterprise Edition Standard Development Kit) dosiahol 2 milióny a 78% vedúcich projektov vidí J2EE ako najefektívnejšiu platformu pre implementáciu a nasadenie WebServices. V roku 2002 J2EE SDK dosiahlo 2 milióny stiahnutí. V ďalšom roku sa používa Java na skoro 550 miliónoch počítačoch a takmer 75% programátorov ju používa ako primárny jazyk na vývoj. V roku 2005 oslavujú Java technológie desiate výročie a používa ju približne 4.5 milióna programátorov.

2.2. Ako to funguje

Java enterprise edition(ďalej len J2EE) je samo o sebe len špecifikácia podľa ktorej sa s pomocou ďalších API špecifikácií programujú enterprise aplikácie, ktoré sú viacvrstvové, škáľovateľné, prenositeľné, znova použiteľné. Teda dá sa povedať že je to súhrn špecifikácií alebo akýchsi guidelines podľa ktorých sa programuje a keď ich spĺňa naprogramovaná aplikácia tak vieme o nej povedať, že je to J2EE aplikácia. J2EE aplikácie musia bežať na aplikačnom servery, ktorý sa za programátora stará o transakcie, bezpečnosť, škálovateľnosť, manažovanie deployovania komponentov. A teda programátor sa môže starať len o samotné programovanie business logiky.

2.3. J2EE API

Základom J2EE sú samozrejme knižnice Java SE(Java standart edition). J2EE k nim ešte pribaľuje ďalšiu funkcionalitu.

- javax.ejb.* Java EnterpriseBeans obsahujú funkcionalitu na podporu perzistencie, RMI(remote procedure call), distribuované objekty, kontrola prístupu,
- javax.transaction.* obsahujú funkcionalitu pre Java Transaction API, ktoré slúži na koordináciu pre viaceré zdroje ako sú napríklad databázy
- javax.xml.stream obsahuje funkcionalitu pre narábanie z XML stream-ami
- javax.jms.* obsahuje funkcionalitu pre Java Messaging Service, ktorý má za úlohu

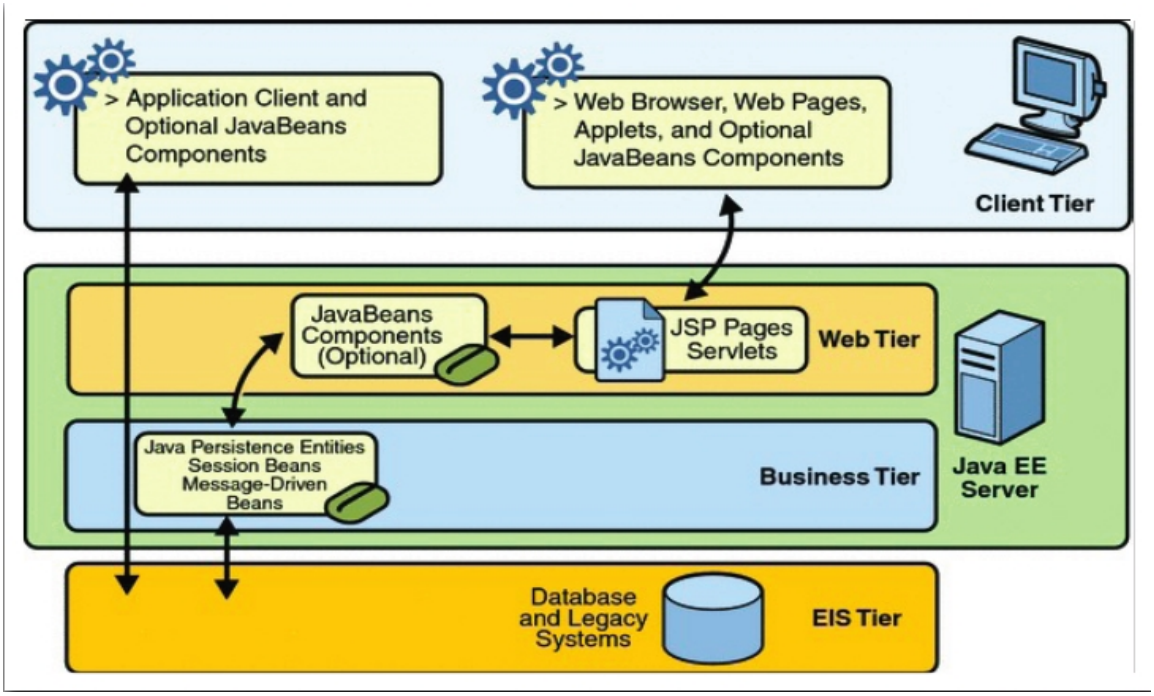
distribuovať správy medzi rôznymi komponentami

– javax.faces.component.html Java Server Faces obsahuje funkcionalitu pre užívateľské rozhrania

– javax.persistence obsahuje Java Persistence API, ktoré sa stará o manažovanie dát

2.4. J2EE komponenty

Komponenta J2EE je samostatná funkčná jednotka, ktorá je súčasťou J2EE aplikácie.



Obrázok 1: J2EE vrstvy

3. Ajax

3.1. História

V deväťdesiatich rokoch 20-teho storočia boli všetky web-stránky statické a teda užívateľská interakcia bola možná len tak že sa celá stránka nahrala znova s nejakým zmenením stavom a teda nebolo možné reagovať na zmenu nejakej jej časti tým že sa iba časť stránky obnovila. Týmto prístupom sa spomaľovala interakcia medzi užívateľom a stránkou.

Riešenie tohto problému bolo v asynchrónnom nahrávaní to znamená možnosť obnovovať časti stránky bez toho aby sa musela opäť nahrávať celá.

Prvé riešenie resp. prvé použiteľné riešenie asynchrónneho nahrávania obsahu bolo pomocou JavaAppletov. JavaApplet je jednoducho povedané skompilovaný Java byte code, ktorý je schopný komunikovať zo serverom.

O rok neskôr v 1996 prišiel IFrame. Pomocou Iframe-u sa dala web stránka rozdeliť na viacej častí. IFrame totiž mohol v sebe držať celý HTML dokument a teda ho aj obnovovať.

Ďalší krok bolo uvedenie XHR(XMLHttpRequest). XHR sa používa na posielanie priamo HTTP alebo HTTPs requestov na web server a spracovanie responsu zo web servera. Dáta prijímané zo servera sú vo forme XML alebo plain text. Tieto dáta sa následne môžu použiť na priamu manipuláciu s DOM objektami bez potreby obnoviť web stránku.

V roku 2005 bolo po prvýkrát predstavené meno AJAX. O rok neskôr vydalo W3C prvú špecifikáciu AJAX-u ako pokus o vytvorenie nového štandardu

3.2. Ako to funguje

Ajax reprezentuje skupinu technológií, ktoré sa štandardne používajú na implementáciu web aplikácií komunikujúcich asynchrónne na pozadí bez toho aby sa musela nahrať celá stránka.

Zoznam technológií

- HTML, XHTML a CSS pre prezentačnú vrstvu
- DOM(Document object model) slúži na dynamické zobrazovanie prenesených dát a interakciu s dátami
- XML, JSON, HTML alebo Plain text sa používa ne prenos dát medzi serverom a

klientom. Resp. na ich zabalenie.

- XHR sa používa na asynchrónnu komunikáciu
- Javascript spája všetky tieto technológie dokopy. Avšak JS nie je jediný skriptovací jazyk ktorý sa dá použiť. Ako alternatíva môže slúžiť napríklad VBScript

Nevýhody

- ťažšie na vývoj ako statické stránky
- stav stránok sa nepamätá automaticky v histórii prehliadača takže po stlačený naspäť v prehliadači sa obvykle dostaneme nie na predchádzajúci stav ale na prvú stránku
- tiež sa nedajú použiť záložky na zapamätanie určitého stavu takejto aplikácie
- takto naprogramované web aplikácie nebudú fungovať v prehliadačoch bez JS a XHR a taktiež v prehliadačoch, ktoré majú tieto funkcie vypnuté
- Ajax aplikácie robia nemalé množstvo requestov na pozadí a sú teda náročnejšie na server.

4. Google web toolkit

4.1. História

GWT(Google web toolkit) je open source java technológia, ktorá umožňuje písať a udržiavať komplexné front-end aplikácie v Jave. Veľká výhoda je v tom ,že programátor nemusí dokonale ovládať technológie okolo AJAX-u ako je Javascript, XMLHttpRequest. Prvá verzia bola vydaná 17.5.2006. Cieľom GWT je aby sa dali pohodlne, rýchlo programovať spoľahlivé, vysoko optimalizované, rýchle web aplikácie. Aj napriek relatívnej mladosti knižnice je veľmi obľúbená a používajú ju tisíce programátorov. Zaujímavé projekte, ktoré boli pomocou GWT naprogramované sú z dielne Google napr. Google Waves, AdWords.

4.2. Ako to funguje

GWT slúži na vyvíjanie AJAX(Asynchrónny Javascript a XML) aplikácií. AJAX je vlastne Javascript na klientskej strane, ktorý asynchrónne komunikuje zo serverom a teda sme schopný interagovať s užívateľom web-u bez toho aby sa nám obnovovala celá stránka. Toto nám umožňuje interagovať s užívateľom na vyššej úrovni. Samotný AJAX má v porovnaný s obyčajným HTML ako výhody tak aj nevýhody.

Jedným nedostatkom AJAX-u je ťažké hľadanie chýb(debugovanie) pre programátora. Práve tento problém alebo aj tento problém rieši GWT.

Pomocou GWT frameworku sa dajú teda jednoducho písať AJAX aplikácie a to nasledovným spôsobom. Nápad ktorý je za týmto spočíva v tom, že GWT funguje akoby dvoma spôsobmi. Prvý sa volá hosted-mode a druhý web-mode.

4.3. Hosted-mode

Hosted-mode je určený na vývoj aplikácie. Pri hosted-mode sa aplikácia nekompiluje do javascriptu ale ostáva celá v jave a to aj klientská strana a samozrejme aj serverová strana. Celá aplikácia navyše beží v takzvanom hosted-browser, ktorý je súčasťou GWT balíčku. To že sa aplikácia v tomto móde neprekladá do javascriptu ale beží ako java byte code nám poskytuje voľnosť v používaný akéhokoľvek nástroja ktorý obsahuje naše vývojové prostredie(IDE).

4.4. Web-mode

Ak máme aplikáciu dokončenú potom ju budeme samozrejme chcieť skompilovať do javascriptu a vyskúšať už na ostro v nejakom prehliadači. Tento krok je taktiež veľmi dôležitý pretože výkon aplikácie v hosted-mode a vo web-mode sa môže líšiť a s pravidla sa aj líši. Vzhľad aplikácie sa taktiež môže líšiť prehliadač od prehliadača pretože GWT využíva ich natívne DOM komponenty. A čo je ešte dôležitejšie je ,že ak používate css tak si treba aplikáciu skontrolovať vo všetkých internetových prehliadačoch. Pokiaľ sa jedná o funkčnosť aplikácie tak ak používate iba komponenty GWT mali by fungovať vo všetkých prehliadačoch ak používate nejaké podporné knižnice alebo ste sami implementovali komponenty tak je potrebné si ich skontrolovať v každom prehliadači.

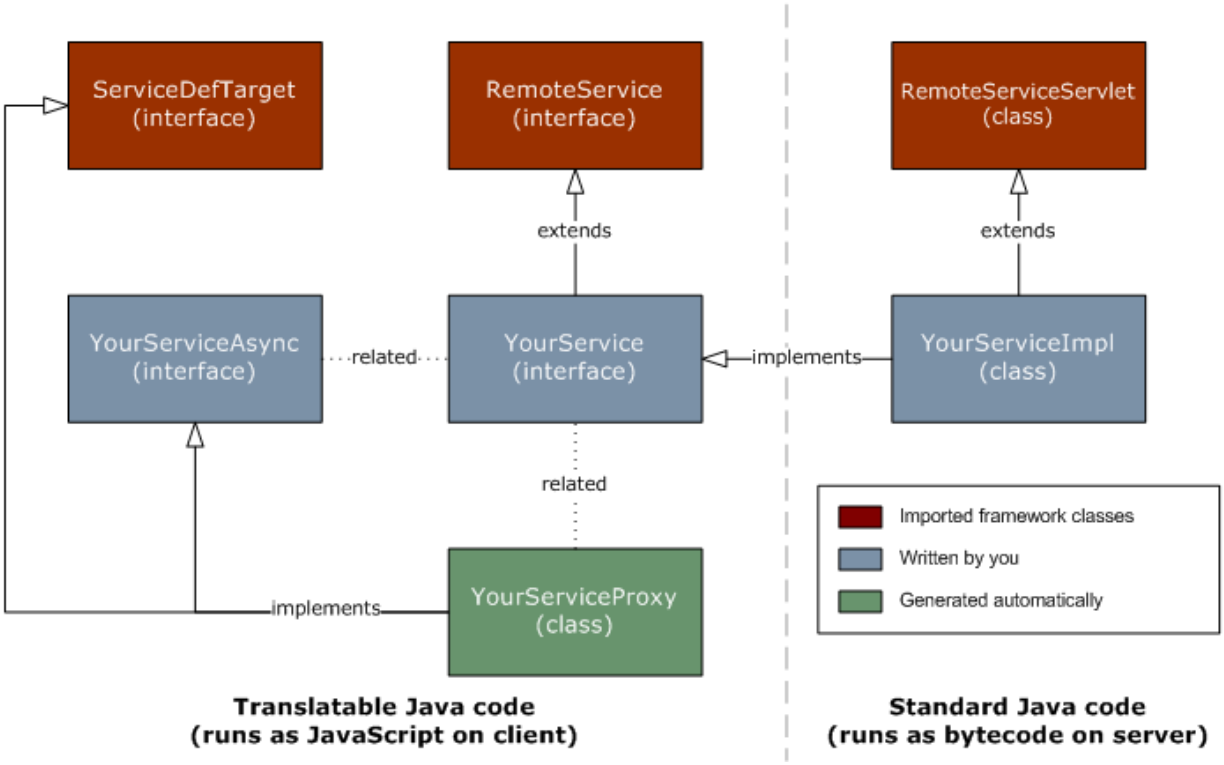
4.5. GWT komponenty

Základné komponenty ktoré nám prináša GWT sú GWT Java-to-Javascript Compiler, ktorý má na starosti prekladanie javy do javascriptu. GWT Hosted web browser sme si už spomínali v kapitole pred. JRE emulation library ktorá slúži na emuláciu najhlavnejších balíčkov z Javy pomocou javascriptu. O tom aké presne balíčky z JRE(Java runtime environment) sa dajú používať. A nakoniec GWT Web UI class library kde sa nachádzajú potrebné triedy a rozhrania na vytváranie GUI komponentov.

4.6. JRE emulation reference

GWT podporuje iba podmnožinu zo všetkých balíčkov javy a navyše pri niektorých triedach podporuje iba podmnožinu metód. Toto je samozrejme dané tým že sila javascriptu nemôže byť porovnávaná s javou samotnou. Ja vypíšem len triedy a metódy ktoré budem potrebovať k implementácii môjho riešenia pre podporu implementácie GWT klienta J2EE aplikácie.

4.7. Komunikácia zo serverom



Obrázok 2: RPC Diagram

Neoddeliteľná časť GWT je samozrejme komunikácia zo serverom. Bez tejto komunikácie by si naše aplikácie nemohli nič zapamätať atď... GWT podporuje protokoly ako JSON a XML ale priamo zabudovaný protokol je GWT RPC(GWT remote procedure call). RPC nám slúži na volanie definovaných metód zo servera do ktorých si môžme posielať argumenty a naopak oni nám posielajú výsledky.

Na obrázku 1 je znázornené ako funguje RPC resp. čo musíme spraviť, implementovať aby sme spravili RPC service. Logicky je to rozdelené na 2 časti a to klientskú, ktorá beží ako Javascript a potom serverovú, ktorá beží ako java bytecode na servery. To čo implementuje programátor je v šedom, ale dve triedy s tých implementovaných sú iba javovské rozhrania, ktoré sú na klientskej strane. Jediný naozajstný kód ktorý treba písať sa nachádza na servery. Teda si treba uvedomiť, že metódy ktoré sa dajú volať sú na strane servera a teda volajú sa metódy zo servera a nie naopak. Naopak sa prenáša len výsledok, ktorý sa samozrejme hneď po obdržaní dá spracovať. Všetko čo sa posiela alebo naopak dostáva ako výsledok v RPC sa musí dať serializovať aby to samotný protokol vedel poslať na server resp. zo

servera. V GWT sú samozrejme pred pripravené typy ktoré sa dajú serializovať ale dajú sa aj definovať nové.

4.8. Pred pripravené serializovateľné typy

- primitívne typy ako : char, byte, short, int, long, boolean, float, double
- inštancie String a Byte. Wrappre primitívnych typov Character, Byte, Short, Integer, Long, Boolean, Float, Double
- enumeration typy
- pole serializovateľných objektov
- serializovateľná užívateľom definovaná trieda
- typ má aspoň jednu serializovateľnú podtriedu

5. GlassFish server

GlassFish je open source aplikačný server pre Java EE platformu. GlassFish je licencovaný duálne pod CDDL(Common Development and Distribution Licence) a GPL(GNU General Public Licence). GlassFish je založený na zdrojovom kóde, ktorý bol darovaný spoločnosťou Sun a Oracle. Používa modifikovanú verziu Apache Tomcat ako kontajner na servlety pre webový obsah s pridanou komponentou, ktorá sa volá Grizzly. Grizzly je komponenta, ktorá používa Java NIO pre rýchlosť a škálovateľnosť.

6. Analýza

6.1. Požiadavky

Čo všetko by mala spĺňať knižnica DataApp2GWTGuiMapper? Najzákladnejšou požiadavkou je samozrejme prenos dát z dátového úložiska, či už je to databáza, XML, súbory, do GUI jeho následné aktualizovanie týmito dátami. Ako to je zrejmé tak musí existovať aj inverzná operácia a to extrahovanie dát z GUI a následné uloženie do databázy.

Ďalšia rozumná požiadavky pri editovaní dát je validácia dát zadaných užívateľom. Validácia dát môže byť veľmi rôznorodá od najjednoduchších ako sú špecifická dĺžka, požadovanie výskytu len niektorých znakov napr. pri telefónnych chceme len čísla, až po zložitejšie a samotným užívateľom definované validácie.

Pre splnenie dvoch vyššie udelených požiadaviek treba navrhnúť systém alebo akési definovanie prepojení medzi dátami a GUI. Teda ktoré dáta sa majú kam zobraziť v GUI a naopak kde sa majú po extrahovaní z GUI uložiť. Taktiež treba definovať ako sa majú jednotlivé GUI komponenty validovať.

Ďalšia vec je volanie akýchsi akcií zo strany servera napríklad akcie ,ktoré sa zavolajú po stlačený tlačidla atď.

Netriviálna požiadavka je aj dobrý návrh celej knižnice aby bolo umožnené pomocou presne definovaných rozhraní doimplementovávať špecifické požiadavky používateľov programátorov.

7. Návrh riešenia

7.1. Idea

Treba si uvedomiť že sa jedná o knižnicu, ktorá bude mať na úlohy ako na strane servera tak aj na strane klienta. Prirodzené riešenie je teda, že knižnica bude mať rozdelenú funkcionalitu pre server a pre klienta.

Funkcionalita na strane servera by mala byť nasledovná :

- inicializácia celej knižnice. Teda zozbieranie potrebných metadát na prepojenie dátovej časti s GUI časťou a zároveň definovanie potrebných validácií, akcií
- triedy na extrahovanie a aktualizovanie dát v nezávislom dátovom úložišti
- dátovú štruktúru ktorou sa budú prenášať dáta na klienta a naspäť
- implementáciu servisov cez ktoré sa budú posielať informácie z klienta na server a naopak
- implementácia servisov na validáciu dát
- implementácia servisov na volanie definovaných akcií

Funkcionalita na strane klienta

- triedy na aktualizáciu dát v GUI a na aktualizáciu dátovej štruktúry pre posielanie dát na server
- dátovú štruktúru, ktorou sa budú prenášať dáta na server a naspäť
- vyvolávanie funkcionality servisov na strane servera

7.2. Funkcionalita na strane servera

7.2.1. Balík dataApp2GWTGuiMapper.actions

Obsahuje triedy, ktoré pracujú s akciami.

1. ActionRepository je úložisko všetkých dostupných akcií, ktoré sú implementované na servery a majú byť dostupné pomocou PerformActionService. Je to singleton.

   a) Membre

   - sharedInstance – zdieľaná statická inštancia

- repository – je mapa kde kľúč je id akcie a hodnota je konkrétna implementácia akcie teda rozhranie Iaction

b) metódy

- public void registerAction(String anId, Iaction anAction) – zaregistruje danú akciu pod daným id. Id akcie musí byť unikátne
- public IAction getItem4Id(String anId) – vráti akciu za dané id

2. IAction je interface, ktorý musí implementovať každá akcia

a) metódy

- public void performAction()

7.2.2. Balík dataApp2GWTGuiMapper.reflection

Obsahuje triedy a rozhrania na pomoc pri práci s reflexiou.

1. ReflectHelper obsahuje metódy na pracú s reflexiou

a) metódy

- public static Object getFieldValue(Object anObject, String aMethodName) – metóda sa pokúša zostrojiť getter s prefixom plus aMethodName a spustiť túto metódu na danom objekte. Je to využívané na získavanie hodnôt z dátových objektov
- public static boolean setFieldValue(Object anObject, String aMethodName, Object aValue) – metóda sa pokúša nasetovať aValue(hodnotu) do daného objektu(anObject) a pomocou danej metódy

2. ReflectionDefs je interface obsahujúci prefixy pre settre a gettre

7.2.3. Balík dataApp2GWTGuiMapper.validation

Obsahuje preddefinované validačné triedy a rozhranie, ktoré implementujú

1. Ivalidation je rozhranie, ktoré musia implementovať všetky validačné triedy

a) metódy

- public boolean validate(Object anObj, Map<String, String> aParameters) – metóda, ktorú musia implementovať všetky validačné triedy. Atribút anObj je v tomto prípade hodnota ktorá ma byť zvalidovaná a v Mape aParameters sa nachádzajú kľúče a hodnoty parametrov pre tú ktorú validačnú triedu. Táto mapa je zostrojená na základe Metadát pre validáciu

7.2.4. Balík dataApp2GWTGuiMapper.metadata.*

V pod balíčkoch sú konkrétne implementácie pre meta dáta a to pre definovanie pomocou XML a anotácií. Programátor má voľné ruky v naprogramovaní iných jemu vyhovujúcich implementácií.

7.2.5. Balík dataApp2GWTGUIMapper.server

Balíček obsahuje triedy, ktoré majú na starosť mapovanie dát z a do dátových tried

Tento balíček obsahuje triedy

1. DataGUIMapperServer je trieda ktorá má na starosť mapovanie dát z a do dátových tried
   a) membre
   b) metódy
      - handleUpdateFormFromObject – má na starosti aktualizáciu FormRepositoryItem. Vytvorí jeho formData pomocou dátových objektov, meta dát a taktiež po vytvára jednotlivé WidgetData pre každý widget
      - hadleUpdateObjectFromForm – má na starosti aktualizáciu dátových tried vo FormRepositoryItem na základe daného formId.

7.2.6. Balík dataApp2GWTGuiMapper.services.server.updateForm

1. UpdateFormServiceImpl je implementácia servisu UpdateFormService na strane servera
   a) membre

b) metódy

- getFormAndMetaData – má jeden vstupný atribút a to formId, ktorý slúži na identifikáciu FormRepositoryItem a pomocou DataGUIMapperServer tento FormRepositoryItem updatne. Metóda vracia množinu obsahujúcu updatnuté FormData a FormMetaData

7.2.7. Balík dataApp2GWTGuiMapper.services.server.updateObject

1. UpdateObjectServiceImpl je implementácia servisu UpdateObjectService na strane servera

   a) membre

   b) metódy

   - public boolean updateData(FormData aData) – má vstupný atribút typu FormData, v ktorom sú čerstvé dáta s klienta a ktoré majú byť uložené do dátových classov. Ak sa operácia podarí tak metóda vráti true.

7.2.8. Balík dataApp2GWTGuiMapper.services.server.validateForm

1. ValidateFormServiceImpl je implementácia servisu ValidateFormService na strane servera

   a) membre

   b) metódy

   - public ValidationDataResult[] validateData(FormData aData) – má za vstup dáta ktoré majú byť zvalidované. Pomocou Metadát vie ktoré validačné metódy a na aké hodnoty má zavolať. Z tohto sa potom vytvorí pole objektov ValidationDataResult v ktorom sú uložené výsledky validácie.

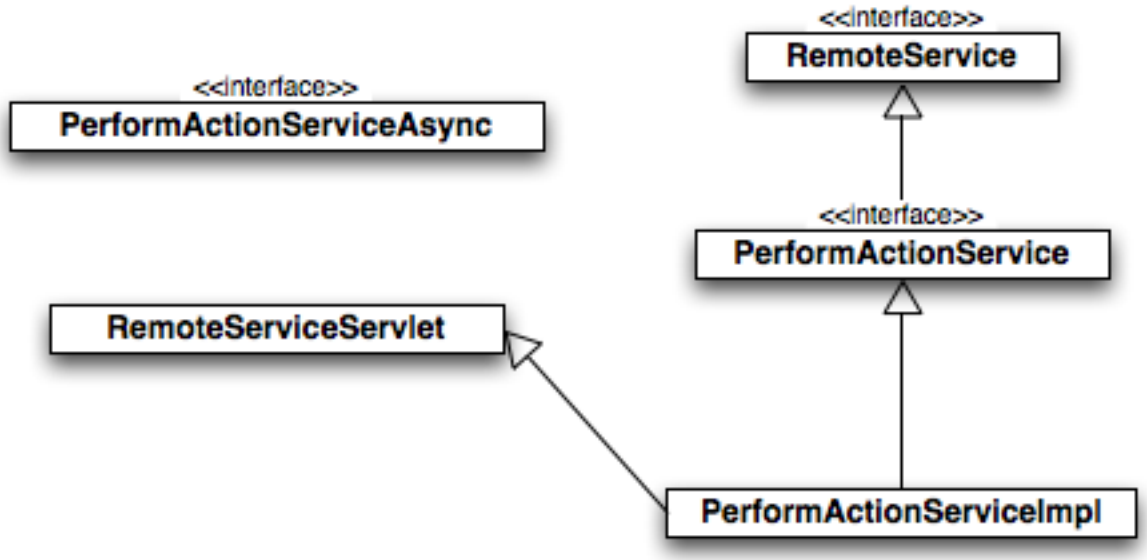
7.2.9. Balík dataApp2GWTGuiMapper.services.server.performAction

1. PerformActionServiceImpl je implementácia servisu PerformActionService na strane servera

a) membre

b) metódy

- public void performAction(String anActionId) – na vstupe dostane id akcie, ktorú má spustiť a ktorá je zaregistrovaná v ActionRepository



Ukážka class diagramu pre servis PerformAction

7.3. Funkcionalita na strane klienta

7.3.1. Balík dataApp2GWTGuiMapper.client

Balíček obsahuje funkcionalitu na mapovanie dát medzi GUI widgetami a triedami na prenos dát na server, validovanie dát, spúšťanie preddefinovaných akcií.

1. DataGUIMapperClient je trieda, ktorá má na starosti mapovanie dát medzi widgetami a triedami na prenos dát na server, validáciu dát, poskytovanie servisov

   a) membre

   - formMetaData – metadáta pre spracovávaný formulár
   - rootPanel – kontajner, ktorý obsahuje všetky widgety vo formulári jeho typ je RootPanel
   - validationResults[] - pole držiace objekty typu ValidationDataResult, ktoré slúžia

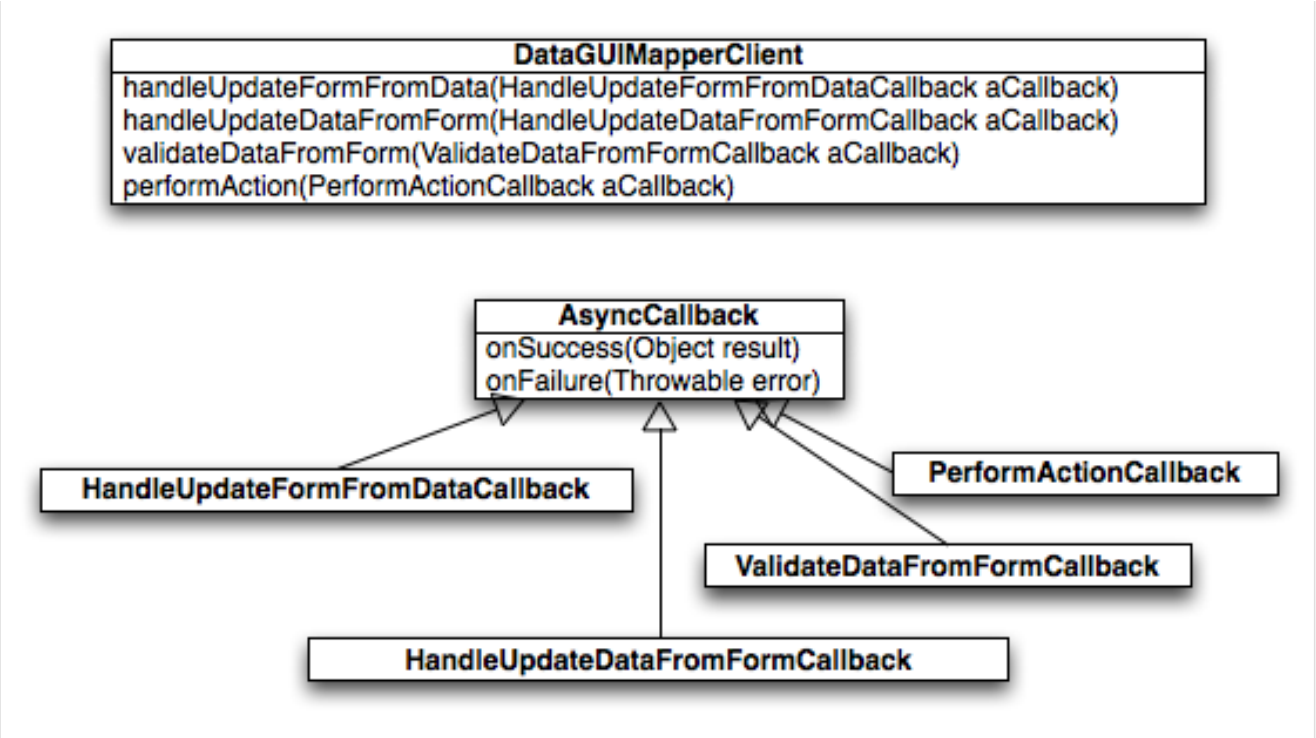
na uloženie výsledkov validácie formulárov

- formId – id formulára. Slúži na identifikáciu formulára resp. dát, metadát k nemu prislúchajúcich na strane servera.

b) metódy

- public void handleUpdateFormFromData() - metóda zavolá servis UpdateFormService a odovzdá jej id formulára, ktorý má byť aktualizovaný a navyše AsyncCallback handleUpdateFormFromDataCallback
- public void handleUpdateDataFromForm() - slúži na zozbieranie dát pre potrebné zaregistrované widget-y a ich následné zaobalenie a odoslanie servisu UpdateObjectService
- public void validateDataFromForm() - slúži na zozbieranie dát, ktoré treba zvalidovať a na ich následné poslanie resp. zavolanie servisu ValidateFormService
- public Widget getWidgetWithName(String aName, Iterator<Widget> aWidgetIterator) – vráti widget s daným menom. Meno widgetu sa momentálne berie z názvu jeho css štýlov je na zváženie či by bolo lepšie brať id alebo dorobiť vlastný tag.
- public void performAction(String anActionId) – zavolá PerformActionService s požadovaným id akcie, ktorá má byť zavolaná
- public static UpdateFormServiceAsync getUpdateFormService() - vytvára asynchrónne volanie UpdateFormService
- public static UpdateObjectServiceAsync getUpdateObjectService() - vytvára asynchrónne volanie UpdateObjectService
- public static ValidateFormServiceAsync getValidateFormService() - vytvára a vracia asynchrónne volanie ValidateFormService
- public static PerformActionServiceAsync getPerformActionService() - vytvára a vracia asynchrónne volanie PerformActionService

- inner class handleUpdateFormFromDataCallback ako už názov naznačuje má na starosť callback teda spracovanie vrátených hodnôt z UpdateFormService. UpdateFormService vracia množinu dvoch prvkov a to FormMetadata a Formdata. FormMetadata sa nadstavia do membra formMetadata a Formdata sa postupne spracujú a teda nadstavia sa do príslušných widgetov tie správne hodnoty.
- inner class handleUpdateDataFromFormCallback nerobí v podstate nič len informuje o tom či bolo zavolanie UpdateDataService úspešné a teda či sa úspešne podarilo aktualizovať dátové štruktúry na strane servera.
- inner class validateDataFromFormCallback nadstaví pomocou metódy addValidationResults() výsledky validácie. Priamo sa nenadstavujú preto lebo ak už je vo výsledkoch validácie rovnaká validácia pre rovnaký widget tak sa táto validácia prepíše a nevyrába sa nová. Týmto je zaistené, že výsledky validácie budú vždy čerstvé a jedinečné.
- inner class performActionCallback iba informuje o tom či sa podarilo akciu úspešne spustiť
- private String getFormId() - nájde id formulára, pre ktorý je táto inštancia DataGUIMapperClient vytvorená
- protected void addValidationResults(ValidationResultData[] aValue) – prechádza už existujúce výsledky validácií a aktualizuje alebo dopĺňa ich.
- public ValidationDataResult[] getValidationDataForWidget(String aWidgetName) – vráti všetky validačné data pre widget s daným menom



Class diagram pre balík client

7.4. Spoločná funkcionalita(Triedy určené na prenos dát)

7.4.1. Balík dataApp2GWTGUIMapper.data

Balíček data obsahuje triedy pomocou, ktorých sa budú prenášať dáta zo servera do klienta a naopak. Ďalej sa tu nachádza funkcionalita na registrovanie inštancií formulárov. Registrácia formulárov je potrebná kvôli neskoršej identifikácii objektov, ktoré sa majú s touto inštanciou nachádzať. Všetky triedy, ktoré majú mať schopnosť prenosu dát resp. tie ktoré sa posielajú medzi klientom a serverom musia byť serializovateľné čo znamená oddedené od rozhrania IsSerializable

Tento balíček obsahuje triedy :

1. WidgetData je trieda na prenos dát z widgetov

   a) membre

   - name – unikátny identifikátor widgetu
   - data – dáta ktoré sa prenášajú či už na server pre uloženie do dátových zdrojov

alebo do klienta kde sa z nich aktualizujú widgety

- multidata – ďalšia forma dát napr. pre kombá

b) metódy – iba settre a gettre pre membre

2. FormData zaobaľuje dáta z formuláru

a) membre

- formId – identifikátor inštancie formuláru, ku ktorému sa vzťahuje práve jeden formulár
- widgetsData – množina WidgetData inštancií

b) metódy

- settre a gettre pre membre
- addWidgetData – pridá WidgetData do množiny widgetsData
- getWidget – vráti widget na základe jej mena

3. FormRepositoryItem trieda ktorá spája meta dáta, dáta na prenášanie a inštancie konkrétnych dátových tried

a) membre

- metadata – meta dáta týkajúce sa daného formuláru
- data – triedy na prenos dát
- dataObjects – inštancie zdrojových dátových tried, ktoré majú byť použité pre tento konkrétny formulár

b) metódy

- settre a gettre pre membre

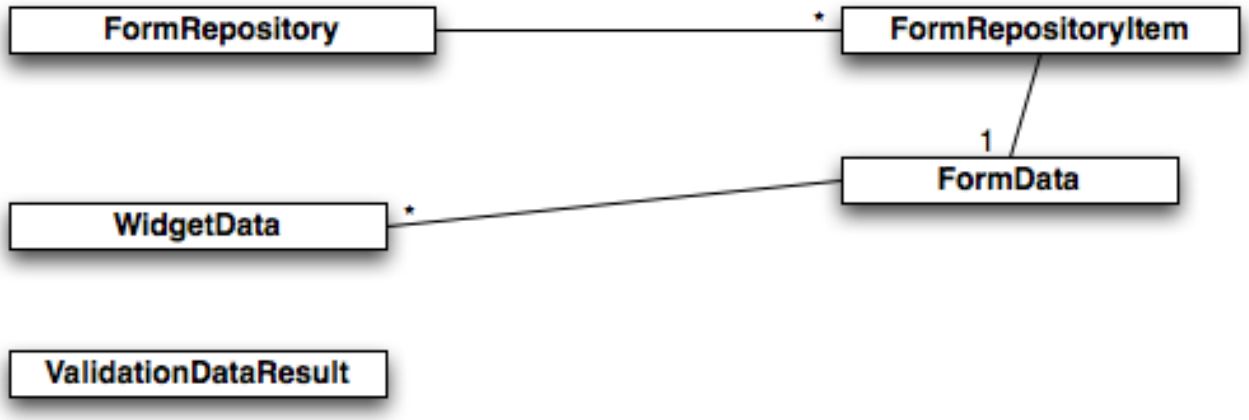
4. FormRepository je singleton ktorý obsahuje všetky registrované FormRepositoryItem

a) membre

- repository – mapa dvojíc kde je ako kľúč identifikátor formuláru a hodnota je inštancia FormRepositoryItem

b) metódy

- getInstance – vráti inštanciu singletona
- registerForm – registruje FormRepositoryItem pod daným formId
- getItem4Id – vráti inštanciu FormRepositoryItem na základe daného formId



Class diagram pre balík data

7.4.2. Balík dataApp2GWTGuiMapper.metadata

V tomto balíku je funkcionalita pre počiatočnú inicializáciu meta dát, ktoré nám budú slúžiť na identifikáciu naviazanosti medzi dátami a GUI a samozrejme aj meta dáta pre určenie validácie hodnôt zadaných užívateľom.

Tento balíček obsahuje triedy :

1. FormMetadata ktorá zodpovedá tagu <form></form> v HTML.

   a) membre

   - name - meno formulára a teda jeho unikátny identifikátor
   - widgets – množina inštancií objektov WidgetMetadata

   b) metódy

   - setName, getName, setWidgets, getWidgets – settre a gettre
   - addWidget – pridá WidgetMetadata do množiny
   - getWidget – vráti meta dáta widgetu na základe daného mena

2. WidgetMetadata ktorá zodpovedá jednotlivým widgetom vo formulároch

   a) membre

      - name – meno widgetu a jeho unikátny identifikátor podľa ktorého sa páruje s widgetom na klientskej strane
      - bindingClass – meno dátovej triedy, ku ktorej sa widget vzťahuje
      - bindingMethod – meno metódy z dátovej triedy, pre ktorú existuje štadardný getter a setter. Z týchto sa potom získavajú a zároveň sa do nich ukladajú dáta
      - validationMetadata – množina meta dát, ktoré určujú ako sa bude validovať widget

   b) metódy

      - setName, getName, setBindingClass, getBindingClass, setBindingMethod, getBindingMethod, setValidationMetadata, getValidationMetadata – settre a gettre
      - addValidation – pridá validáciu pre widget

3. ValidationMetadata obsahuje meta dáta pre validovanie dát z widgetov.

   a) membre

      - validationClassName – meno validačnej triedy
      - validationKeys – pole kľúčov. Napr. MaxLength pri určení maximálnej dĺžky reťazca
      - validationValues – pole hodnôt týchto kľúčov
      - isClientSideValidation – boolean hodnota, ktorá určuje či má byť validácia vykonaná na strane klienta. Ak je false tak musia poslať všetky informácie o validácii na server a tam sa spracovať

   b) metódy

      - iba settre a gettre pre membre

4. IRegister je rozhranie kde sa registrujú objekty z ktorých budú meta dáta extrahované.

Keďže je to iba rozhranie tak je potrebná konkrétna implementácia, ktorá závisí od toho pre akú reprezentáciu sa programátor rozhodne

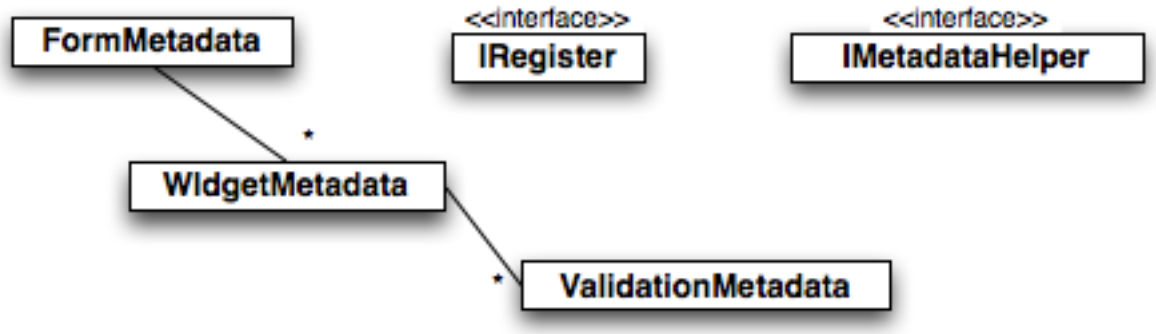
a) metódy

- registerObject – má za úlohu registrovať objekt obsahujúci meta dáta
- getRegisteredObjects – vráti množinu registrovaných objektov obsahujúce meta dáta

5. IMetadataHelper je rozhranie ktoré treba implementovať na základe reprezentácie meta dát

a) metódy

- buildFormsMetadata – implementácia tejto metódy má za úlohu zozbierať všetky meta dáta a pripraviť FormMetadata, WidgetMetadata, ValidationMetadata objekty pre neskoršie použitie



Class diagram pre balík metadata

7.5. Návrh anotácií – modelu metadát

V tejto kapitole definujem štruktúru anotácií a teda model meta dát, ktorými sa budú definovať prepojenia medzi GUI a dátami a validácia dát. Implementácia anotácií sa nachádza v balíku dataApp2GWTGuiMapper.metadata.annotations

Nachádzajú sa tu Java annotation type. Sú to definície rôznych anotácií, ktoré sa budú používať pri definovaný či už vzťahov medzi widgetami a dátami tak aj definovanie validácií. Poďme sa teda bližšie pozrieť na tieto anotácie. Všetky mnou definované anotácie budú ešte

navyše anotované a to nasledovné @Retention(RetentionPolicy.RUNTIME) čo znamená, že sa tieto anotácie sú prístupné za behu programu pomocou reflexie. A ešte @Target(ElementType.METHOD) čo znamená, že anotácie sú aplikovateľné len pre metódy.

1. Form
    a) membre
        - name – názov formulára ku ktorému sa meta dáta viažu
        - widget – anotácia widget bude definovaná nižšie
2. Widget
    a) membre
        - name – názov widgetu ku ktorému sa meta dáta viažu. Widget musí byť vždy použitý len vo vnútri Form inak nebude spracovaný
3. Forms - keďže objekt nemôže byť anotovaný tou istou anotáciou dva krát bola nutnosť to týmto spôsobom. Dôvod je jednoduchý jedny dáta sa môžu nachádzať vo viacerých formulároch.
    a) membre
        - forms – pole objektov typu Form.
4. Validation – prázdna anotácia ktorá slúži ako rodičovská anotácia pre všetky Validačné anotácie.
5. MinLength – anotovaná navyše s @Validation. Definuje minimálnu dĺžku vstupu
    a) membre
        - value – min dĺžka vstupu
6. MaxLength – anotovaná navyše s @Validation. Definuje maximálnu dĺžku vstupu
    a) membre
        - value – max dĺžka vstupu
7. Pattern – anotovaná navyše s @Validation. Definuje pattern vstupu

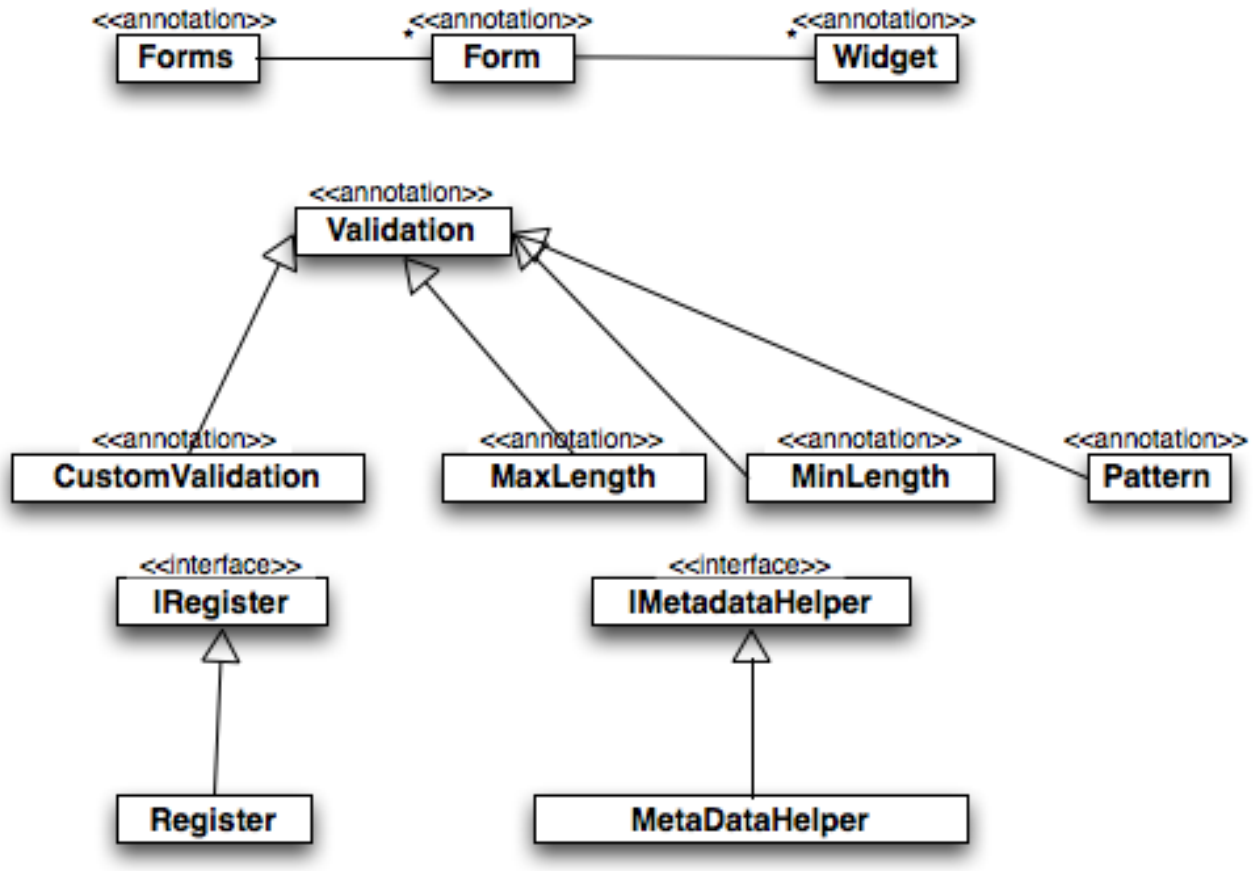
a) membre

- regex – regulárny výraz voči ktorému bude vstup kontrolovaný

8. CustomValidation – anotovaná navyše @Validation. Definuje užívateľom definovanú validáciu

a) membre

- className – meno triedy, ktorá bude použitá na validáciu
- methodName – meno metódy y z triedy className, ktorá bude použitá na validáciu
- arguments – pole Class objektov argumentov pre metódu validácie



Class diagram pre metadata

7.6. Ako to celé bude fungovať

V tejto kapitole sa pokúsim vysvetliť ako bude celá knižnica fungovať. Akým spôsobom sa knižnica má používať.

Programátor, ktorý bude chcieť použiť knižnicu bude musieť ako prvý krok dať skompilovanú DataApp2GWTGuiMapper.jar do classpathu svojej aplikácie. Keďže GWT má knižnice na báze modulov, ktoré sú definované pomocou XML súborov je treba každý GWT modul oddediť od DataGUIMapperClient tak ako je vidno na obrázku 3 v riadku <inherits name =”dataApp2GWTGuiMapper.DataGUIMapperClient”>

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
    <inherits name="com.google.gwt.user.User"/>
    <inherits name="dataApp2GWTGuiMapper.DataGUIMapperClient"/>
    <source path="test"/>
    <source path="client"/>
    <entry-point class="org.milan.client.bakalarkaEntryPoint"/>
    <!-- Do not define servlets here, use web.xml -->
</module>
```

Obrázok 3: Oddedenie modulu

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
3      <servlet>
4          <servlet-name>UpdateFormService</servlet-name>
5          <servlet-class>dataApp2GWTGuiMapper.services.server.updateForm.UpdateFormServiceImpl</servlet-class>
6      </servlet>
7      <servlet-mapping>
8          <servlet-name>UpdateFormService</servlet-name>
9          <url-pattern>/org.milan.bakalarka/updateform/updateformservice</url-pattern>
10     </servlet-mapping>
11     <servlet>
12         <servlet-name>UpdateObjectService</servlet-name>
13         <servlet-class>dataApp2GWTGuiMapper.services.server.updateObject.UpdateObjectServiceImpl</servlet-class>
14     </servlet>
15     <servlet-mapping>
16         <servlet-name>UpdateObjectService</servlet-name>
17         <url-pattern>/org.milan.bakalarka/updateobject/updateobjectservice</url-pattern>
18     </servlet-mapping>
```

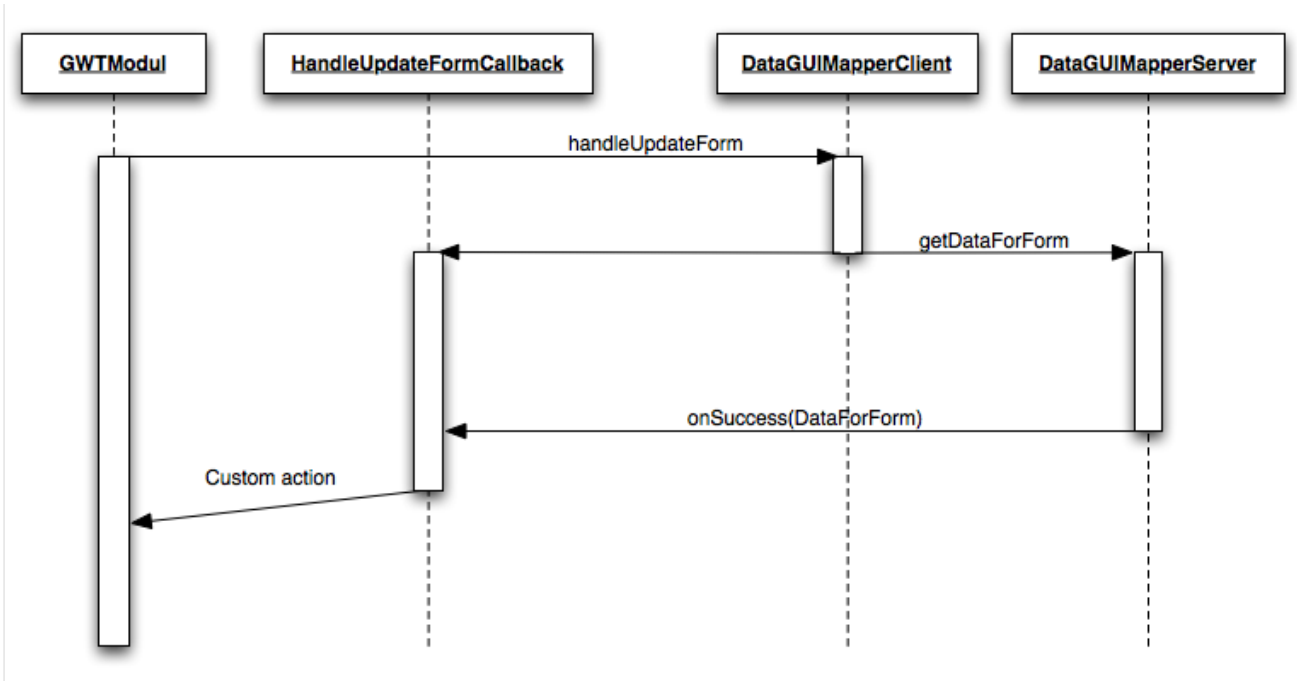
Obrázok 4: Definovanie potrebných servisov

Po tomto kroku je treba sprístupniť servisy, cez ktoré knižnica komunikuje zo serverom. Na obrázku 4 je vidno definovanie pomocou XML. Od riadku 4 po riadok 18 z obrázku je definícia potrebných servisov. Tag <servlet> obsahuje názov servisu a triedu kde je implementovaný. Tag <servlet-mapping> obsahuje názov servisu a jeho URL. Cast URL v tomto prípade org.milan.bakalarka musí byť vymenená za cestu k XML ktoré definuje modul ktorý chce používať tieto servisy.

Po naprogramovaní užívateľského interfacu pomocou GWT je treba do tohto modulu pridať member DataGUIMapperClient. Tento má jediný konštruktor v ktorom prijíma triedu typu Panel, od ktorej sú v GWT oddedené všetky kontajnery na držanie widgetov. Teda panel, ktorý obsahuje všetky GWT komponenty. Pri inicializácii teda pri nahrávaní nášho GWT modulu treba skonštruovať unikátne id pre tento panel najjednoduchšie a odporúčané je použiť System.currentTimeInMillis() metódu. Cez toto id sa registruje Panel k určitému dátovému objektu.

Ďalší krok je spojazdnenie aktualizácie dát Widgetov. Tu stačí zavolať metódu z DataGUIMapperClienta a to handleUpdateFormFromData. Táto metóda by mala byť zavolaná po inicializácii celého GUI v DEMO aplikácii je zavolaná na metódy load().

Validácia funguje nasledovne. Z ľubovolných GWT handlerov či už na onCLick, onChange atď... sa bude volať metóda opäť z DataGUIMapperClient. Konkrétne je to metóda validateDataFromForm(), ktorá prijíma na vstupe AsyncCallback aby si užívateľ mohol výsledky validácie spracovať podľa vlastného uváženia. Pre tento AsyncCallback je vytvorená rodičovská trieda ValidateDataFromFormCallback. Túto triedu treba by vytváraný oddediť a do overridnutých metód netreba zabudnúť na volanie metód z rodiča.



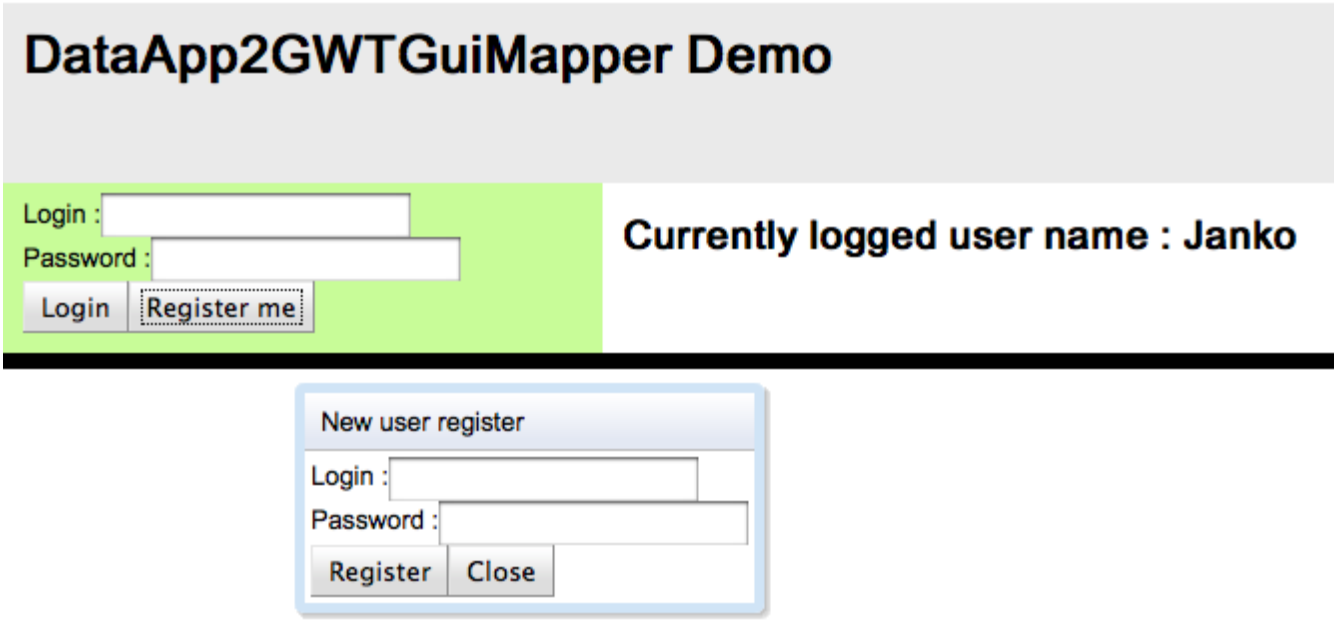
Sequence diagram pre cyklus servisu

Volanie akcií funguje presne tak isto ako validácia z rozdielom toho ze AsyncCallback od

ktorého treba dediť sa volá PerformActionCallback. Výsledky akcií sa zatiaľ vracajú v podobe mapy Stringov.

8. Demo aplikácia

Pre demonštráciu vytvoreného frameworku som vytvoril jednoduchú web aplikáciu, v ktorej sa dá zaregistrovať a následne prihlásiť. Registračné údaje sa uložia do databázy. Ako databázu som použil PostgreSQL.



Obrázok 5: Screenshot z demo aplikácie

9. Vývoj knižnice

9.1. Stanovenie cieľov implementácie

Cieľom je naimplementovať kostru frameworku. Čo zahŕňa kompletnú analýzu požiadaviek na framework. Následné premyslenie a rozdelenie funkcionality do vhodných logických celkov. Návrh dátových štruktúr na prenos dát medzi serverom a klientom.

Návrh a implementácia aspoň jedného spôsobu identifikácie dát, validácie buď pomocou anotácií alebo XML.

Návrh a implementácia validovania dát na servery a taktiež spúšťanie akcií na servery.

9.2. Čo sa podarilo implementovať

Implementovaná identifikácia dát, validácií pomocou anotácií.

Podarilo sa implementovať aktualizáciu dát vo formulároch a taktiež ich prenos naspať na server. Framework podporuje iba pár widgetov na ukážku a to Label, PopUp a TextField

Implementované akcie a validácie na strane servera.

9.3. Čo treba ešte doimplementovať

Frameworku chýba k úplnosti implementácia všetkých dostupných widgetov z GWT frameworku ako sú Table atď...

Ďalej chýba možnosť rozširovať framework o vlastné typy validácií

10. Použitá literatúra

[1] Ajax - http://en.wikipedia.org/wiki/Ajax_(programming)

[2] Java Applet - http://en.wikipedia.org/wiki/Java_applet

[3] IFrame - http://en.wikipedia.org/wiki/HTML_element#Frames

[4] DOM - http://en.wikipedia.org/wiki/Document_Object_Model

[5] XMLHttpRequest - http://en.wikipedia.org/wiki/XMLHttpRequest

[6] History of Java - http://www.java.com/en/javahistory/timeline.jsp

[7] GWT - http://code.google.com/webtoolkit/

[8] Anotácie - http://java.sun.com/developer/technicalArticles/J2SE/constraints/annotations.html

[9] W3C - http://www.w3.org/

[10] JSON - http://www.json.org/

[11] VBScript - http://en.wikipedia.org/wiki/VBScript

[12] Google Waves - http://en.wikipedia.org/wiki/Google_Wave

[13] AdWords - http://en.wikipedia.org/wiki/AdWords

[14] Glassfish - https://glassfish.dev.java.net/

11. Prílohy

K práci je priložené CD so zdrojovými kódmi frameworku a DEMO aplikáciou